English

FUJITSU Software

openUTM V6.4

Administering Applications

User Guide

Edition July 2016

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:
manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2008

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Copyright © 2016 Fujitsu Technology Solutions GmbH.

All rights reserved.
Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

Contents

1 Preface . . . 13

1.1 Summary of contents and target group . . . 15

1.2 Summary of contents of the openUTM documentation . . . 16
1.2.1 openUTM documentation . . . 16
1.2.2 Documentation for the openSEAS product environment . . . 21
1.2.3 Readme files . . . 22

1.3 Innovations in openUTM V6.4 . . . 23
1.3.1 New server functions . . . 23
1.3.2 New client functions . . . 26
1.3.3 New and modified functions for openUTM WinAdmin . . . 27
1.3.4 New functions for openUTM WebAdmin . . . 27

1.4 Notational conventions . . . 29

2 Overview of openUTM administration . . . 31

2.1 Command interface . . . 34

2.2 KDCADMI program interface . . . 37

2.3 Sample programs . . . 41

2.4 PADM, DADM for administering message queues and printers . . . 42

2.5 Administration tool CALLUTM . . . 42

2.6 openUTM WinAdmin and openUTM WebAdmin . . . 43

3 Administering objects and setting parameters . . . 45

3.1 Information functions in openUTM . . . 46

3.2 Performance check . . . 48
3.2.1 Information about the utilization level of the application . . . 48
3.2.2 Diagnosing errors and bottlenecks . . . 50
3.2.3 Possible measures . . . 51

3.3 Avoiding a page pool bottleneck . . . 58
3.3.1 Page pool of a standalone application . . . 58
3.3.2 Page pools of a UTM cluster application . . . 62

3.4 Replacing the application program . . . 63

3.5 Clients and printers . . . 64

4 Changing the configuration dynamically . . . 67

4.1 Requirements for KDCDEF generation . . . 69

4.2 Adding objects to the configuration dynamically . . . 73
4.2.1 Adding clients, printers and LTERM partners . . . 74
4.2.2 Adding program units, transaction codes, TAQ queues and VORGANG exits . . . 78
4.2.3 Creating user IDs . . . 79
4.2.4 Creating key sets . . . 79
4.2.5 Entering LU6.1 connections for distributed processing . . . 80
4.2.6 Entering LTACs . . . 81
4.2.7 Format and uniqueness of object names . . . 82

4.3 Deleting objects dynamically from the configuration . . . 85
4.3.1 Deleting clients/printers and LTERM partners . . . 87
4.3.2 Deleting program units, transaction codes and VORGANG exits . . . 89
4.3.3 Deleting user IDs . . . 91
4.3.4 Deleting key sets . . . 93
4.3.5 Deleting LU6.1 connections and sessions . . . 93
4.3.6 Deleting LTACs . . . 94

4.4 Modifying object properties . . . 95
4.4.1 Modifying clients/printers and LTERM partners . . . 96
4.4.2 Modifying transaction codes and TAC queues . . . 97
4.4.3 Modifying user IDs . . . 98
4.4.4 Modifying key sets . . . 98
4.4.5 Modifying LU6.1 sessions . . . 99

5 Generating KDCDEF statements from the KDCFILE . . . 101

5.1 Starting the inverse KDCDEF . . . 103

5.2 Result of the inverse KDCDEF run . . . 105

5.3 Inverse KDCDEF for version migrations . . . 106

5.4 Recommendations for regeneration of an application . . . 107

6 Administration using commands . . . 109

6.1 Administration in dialog . . . 110

6.2 Administration using message queuing . . . 112

7 Writing your own administration programs . . . 115

7.1 Dialog administration programs . . . 116
7.1.1 Several administration calls . . . 116
7.1.2 Multi-step service . . . 117

7.2 Diagnostic options for the administration interface . . . 119

8 Central administration of several applications . . . 121

8.1 Administration using WinAdmin and WebAdmin . . . 123
8.1.1 Adapting generation of the UTM application . . . 124
8.1.2 Configuration of WinAdmin and WebAdmin . . . 125

8.2 Configuration models for own application of administration . . . 127
8.2.1 Administration via UPIC clients . . . 128
8.2.2 Administration via distributed processing . . . 133
8.2.3 Administration via a TS application . . . 138

8.3 Central Administration using commands . . . 140

8.4 Central Administration using programs . . . 141
8.4.1 Decentralized administration programs . . . 141
8.4.2 Central administration programs . . . 144

9 Automatic administration . . . 147

9.1 Control using the MSGTAC program . . . 148

9.2 Control via user-specific message destinations . . . 151

10 Access rights and data access control . . . 153

10.1 Configuring the administrator connection . . . 155

10.2 Granting administration privileges . . . 156

10.3 Generating administration commands . . . 157

11 Program interface for administration - KDCADMI . . . 159

11.1 Calling the KDCADMI functions . . . 160
11.1.1 The KDCADMI function call . . . 161
11.1.2 Description of the data areas to be supplied . . . 162
11.1.3 Return codes . . . 174
11.1.4 Supplying the fields of the data structure with data when passing data . . . 177

11.2 KDCADMI operation codes . . . 178
11.2.1 KC_CHANGE_APPLICATION- Change application program . . . 179
11.2.2 KC_CREATE_DUMP - Create a UTM dump . . . 186
11.2.3 KC_CREATE_OBJECT - Add objects to the configuration . . . 188
11.2.3.1 obj_type=KC_CON . . . 193
11.2.3.2 obj_type=KC_KSET . . . 195
11.2.3.3 obj_type=KC_LSES . . . 196
11.2.3.4 obj_type=KC_LTAC . . . 197
11.2.3.5 obj_type=KC_LTERM . . . 200
11.2.3.6 obj_type=KC_PROGRAM . . . 205
11.2.3.7 obj_type=KC_PTERM . . . 207
11.2.3.8 obj_type=KC_TAC . . . 214
11.2.3.9 obj_type=KC_USER . . . 222
11.2.3.10 Returncodes . . . 230
11.2.4 KC_CREATE_STATEMENTS - Create KDCDEF control statements (inverse KDCDEF) . . . 246
11.2.5 KC_DELETE_OBJECT - Delete objects . . . 256
11.2.6 KC_ENCRYPT - Create, delete, read RSA key pairs . . . 268
11.2.7 KC_GET_OBJECT - Query information . . . 278
11.2.8 KC_LOCK_MGMT - Release locks in UTM cluster applications . . . 306

11.2.9 KC_MODIFY_OBJECT - Modify object properties and application parameters 311
11.2.9.1 obj_type=KC_CLUSTER_NODE 320
11.2.9.2 obj_type=KC_KSET 322
11.2.9.3 obj_type=KC_LOAD_MODULE 323
11.2.9.4 obj_type=KC_LPAP 325
11.2.9.5 obj_type=KC_LSES 328
11.2.9.6 obj_type=KC_LTAC 330
11.2.9.7 obj_type=KC_LTERM 332
11.2.9.8 obj_type=KC_MUX 336
11.2.9.9 obj_type=KC_OSI_CON 338
11.2.9.10 obj_type=KC_OSI_LPAP 339
11.2.9.11 obj_type=KC_PTERM 343
11.2.9.12 obj_type=KC_TAC 347
11.2.9.13 obj_type=KC_TACCLASS 351
11.2.9.14 obj_type=KC_TPOOL 355
11.2.9.15 obj_type=KC_USER 357
11.2.9.16 obj_type=KC_CLUSTER_CURR_PAR 362
11.2.9.17 obj_type=KC_CLUSTER_PAR 363
11.2.9.18 obj_type=KC_CURR_PAR 365
11.2.9.19 obj_type=KC_DIAG_AND_ACCOUNT_PAR 369
11.2.9.20 obj_type=KC_MAX_PAR 377
11.2.9.21 obj_type=KC_TASKS_PAR 380
11.2.9.22 obj_type=KC_TIMER_PAR 382
11.2.9.23 Return codes 385
11.2.10 KC_ONLINE_IMPORT - Import application data online 398
11.2.11 KC_PTC_TA - Roll back transaction in PTC state 401
11.2.12 KC_SEND_MESSAGE - Send message (BS2000 systems) 404
11.2.13 KC_SHUTDOWN - Terminate the application run 409
11.2.14 KC_SPOOLOUT - Establish connections to printers 418
11.2.15 KC_SYSLOG - Administer the system log file 422
11.2.16 KC_UPDATE_IPADDR - Update IP addresses 433
11.2.17 KC_USLOG - Administer the user log file 440

11.3 Data structures used to pass information 443
11.3.1 Data structures for describing object properties 445
kc_abstract_syntax_str - Abstract syntax for communication svia OSI TP 446
kc_access_point_str - OSI TP access point 447
kc_application_context_str - Application context for communication via OSI TP . . . 453
kc_bcamappl_str - Names and addresses of the local application 454
kc_cluster_node_str - Node applications of a UTM cluster application 457
kc_con_str - LU6.1 connections 462
kc_db_info_str - Output database information 466
kc_edit_str - EDIT profile options (BS2000 systems) 468
kc_gssb_str - Global secondary storage areas of the application 472

kc_kset_str - Key sets of the application . . . 473
kc_load_module_str - Load modules (BS2000 systems) or shared objects/DLLs (Unix, Linux and Windows systems) . . . 475
kc_lpap_str - Properties of LU6.1 partner applications . . . 478
kc_lses_str - LU6.1 sessions . . . 484
kc_ltac_str - Transaction codes of remote services (LTAC) . . . 487
kc_lterm_str - LTERM partners . . . 492
kc_message_module_str - User message modules . . . 503
kc_mux_str - Multiplex connections (BS2000 systems) . . . 505
kc_osi_association_str - Associations to OSI TP partner applications . . . 509
kc_osi_con_str - OSI TP connections . . . 511
kc_osi_lpap_str - Properties of OSI TP partner applications . . . 517
kc_program_str - Program units and VORGANG exits . . . 523
kc_ptc_str - Transactions in PTC state . . . 527
kc_pterm_str - Clients and printers . . . 529
kc_queue_str - Properties of temporary queues . . . 543
kc_sfunc_str - Function keys . . . 544
kc_subnet_str - Information on subnets . . . 546
kc_tac_str - Transaction codes of local services . . . 547
kc_tacclass_str - TAC classes for the application . . . 559
kc_tpool_str - LTERM pools for the application . . . 562
kc_transfer_syntax_str - Transfer syntax for communication via OSI TP . . . 572
kc_user_str, kc_user_fix_str, kc_user_dyn1_str and kc_user_dyn2_str user IDs . . . 573
11.3.2 Data structures used to describe the application parameters . . . 591
kc_cluster_curr_par_str - Statistics values of a UTM cluster application . . . 592
kc_cluster_par_str - Global properties of a UTM cluster application . . . 593
kc_curr_par_str - Current values of the application parameters . . . 600
kc_diag_and_account_par_str - Diagnostic and accounting parameters . . . 610
kc_dyn_par_str - Dynamic objects . . . 619
kc_max_par_str - Maximum values for the application (MAX parameters) . . . 624
kc_msg_dest_par_str - Properties of the user-specific message destinations . . . 640
kc_pagepool_str - Current utilization of the page pool . . . 641
kc_queue_par_str - Properties of queue objects . . . 643
kc_signon_str - Properties of the sign-on process . . . 644
kc_system_par_str - System parameters . . . 648
kc_tasks_par_str - Number of processes . . . 652
kc_timer_par_str - Timer settings . . . 656
kc_utmd_par_str - Parameters for distributed processing . . . 661

12 Administration commands - KDCADM 663

KDCAPPL - Change properties and limit values for an operation 666
KDCBNDL - Replace Master LTERM 680
KDCDIAG - Switch diagnostic aids on and off 681
KDCHELP - Query the syntax of administration commands 690
KDCINF - Request information on objects and application parameters 691
KDCLOG - Change the user log file 742
KDCLPAP - Administer connections to (OSI-)LPAP partners 743
KDCLSES - Establish/shut down connections for LU6.1 sessions 751
KDCLTAC - Change the properties of LTACs 753
KDCLTERM - Change the properties of LTERM partners 755
KDCMUX - Change properties of multiplex connections (BS2000 systems) 758
KDCPOOL - Administer LTERM pools 762
KDCPROG - Replace load modules/shared objects/DLLs 764
KDCPTERM - Change properties of clients and printers 769
KDCSEND - Send a message to LTERM partners (BS2000 systems) 774
KDCSHUT - Terminate an application run 775
KDCSLOG - Administer the SYSLOG file 779
KDCSWTCH - Change the assignment of clients and printers to LTERM partners . . 785
KDCTAC - Lock/release transaction codes and TAC queues 789
KDCTCL - Change number of processes of a TAC class 792
KDCUSER - Change user properties 797

13 Administering message queues and controlling printers 799

13.1 Authorizations concept (BS2000, Unix and Linux systems) 802

13.2 Administering message queues (DADM) 805
13.2.1 Displaying information on messages in a queue - DADM RQ 808
13.2.2 Reading user information about a message - DADM UI 809
13.2.3 Prioritizing messages in the queue - DADM CS 809
13.2.4 Deleting messages from a queue - DADM DA/DL 810
13.2.5 Move messages from the dead letter queue - DADM MA/MV 811

13.3 Administering printers and control print output (PADM) 812
13.3.1 Administering printers with PADM 812
13.3.1.1 Querying information about a printer PADM PI 813
13.3.1.2 Changing the printer status - PADM CS 814
13.3.1.3 Assigning a printer to another LTERM partner - PADM CA 814

13.3.2 Print control with PADM . . . 815
13.3.2.1 Activating/deactivating confirmation mode - PADM AC/AT . . . 817
13.3.2.2 Confirming or repeating print output - PADM OK/PR . . . 818
13.3.2.3 Querying information about print jobs to be confirmed - PADM AI . . . 818
13.3.3 Handling of errors during print output . . . 819

13.4 UTM program units for DADM and PADM functions . . . 820
13.4.1 Generating KDCDADM and KDCPADM . . . 820
13.4.2 KDCDADM - Administer messages . . . 821
DELETE - Delete messages from the message queue . . . 822
INFORM - Display information about message queues and messages . . . 824
MOVE - Move messages from the dead letter queue . . . 827
NEXT - Prioritize messages in the message queue . . . 829
13.4.3 KDCPADM - Print control and printer administration . . . 830
INFORM - Display information about printers for a printer control LTERM . . . 831
MODE - Change the confirmation mode for a printer . . . 835
PRINT - Confirm / repeat print job . . . 836
STATE - Change the status of a printer . . . 837
SWITCH - Change the assignment of printers to LTERM partners . . . 838

14 Appendix . . . 839

14.1 Program interface for administration in COBOL . . . 839
14.1.1 COPY members for the program interface in COBOL . . . 840
14.1.2 KDCADMI function call . . . 842
14.1.3 Notes on programming . . . 843

14.2 Sample programs . . . 844
14.2.1 The C program units ADJTCLT . . . 847

14.3 CALLUTM - Tool for administration and client/server communication . . . 853
14.3.1 Instructions for use . . . 854
14.3.1.1 Generation . . . 854
14.3.2 Description of program statements . . . 860
14.3.3 Components, system environment, software configuration . . . 876
14.3.4 Integration in a UTM application . . . 876
14.3.5 Execution . . . 876
14.3.6 Program-monitoring job variables . . . 877
14.3.7 Messages issued by CALLUTM . . . 878

Glossary 881

Abbreviations 917

Related publications 923

Index 933

1 Preface

Modern enterprise-wide IT environments are subjected to many challenges of rapidly increasing importance. This is the result of:

- heterogeneous system landscapes
- different hardware platforms
- different networks and different types of network access (TCP/IP, SNA, ...)
- the applications used by companies

Consequently, problems arise – whether as a result of mergers, joint ventures or labor-saving measures. Companies are demanding flexible, scalable applications, as well as transaction processing capability for processes and data, while business processes are becoming more and more complex. The growth of globalization means, of course, that applications are expected to run 24 hours a day, seven days a week, and must offer high availability in order to enable Internet access to existing applications across time zones.

openUTM is a high-end platform for transaction processing that offers a runtime environment that meets all these requirements of modern, business-critical applications, because openUTM combines all the standards and advantages of transaction monitor middleware platforms and message queuing systems:

- consistency of data and processing
- high availability of the applications (not just the hardware)
- high throughput even when there are large numbers of users (i.e. highly scalable)
- flexibility as regards changes to and adaptation of the IT system

An UTM application can be run as a standalone UTM application or sumultanously on several different computers as a UTM cluster application.

openUTM forms part of the comprehensive **openSEAS** offering. In conjunction with the Oracle Fusion middleware, openSEAS delivers all the functions required for application innovation and modern application development. Innovative products use the sophisticated technology of openUTM in the context of the **openSEAS** product offering:

- BeanConnect is an adapter that conforms to the Java EE Connector Architecture (JCA) and supports standardized connection of UTM applications to Java EE application servers. This makes it possible to integrate tried-and-tested legacy applications in new business processes.
- The WebTransactions member of the openSEAS family is a product that allows tried-and-tested host applications to be used flexibly in new business processes and modern application scenarios. Existing UTM applications can be migrated to the Web without modification.

1.1 Summary of contents and target group

The manual “Administering Applications” is intended for UTM application administrators and administration programmers. It describes the program interface for administration which you can use to write your own administration programs, the administration command interface, and the options available for administering message queues.

Readers are expected to have a thorough grasp of the C programming language and to be familiar with openUTM. It is particularly important to have competent knowledge of the generation tool KDCDEF and the program interface KDCS. For further information, please refer also to the openUTM manuals “Generating Applications” and “Programming Applications with KDCS”.

Chapters 2, 3, 8, 9 and 10 of this manual contain general information about UTM administration. They are intended both for programmers who write their own administration programs and for the users who use the administration programs. For example, they provide information on the various interfaces that openUTM offers for administering your UTM application, contain examples of how you can use the openUTM administration functions to ensure that your application offers lasting performance and reliability, and introduce you to the options available for central and automatic administration. Chapter 8 also examines the administration of UTM cluster applications in greater detail.

Chapters 4, 5, 7 and 11 contain special information for programmers who write their own administration programs. They provide a detailed description of the structure of administration programs and of the dynamic entry and deletion of clients, printers, services and user IDs. Chapter 11 contains all the administration calls for the C program interface and the C data structures of the interface. It also describes in detail which administration functions you can implement with the aid of the interface.

Chapters 6 and 12 address the particular needs of the users of administration commands. Chapter 6 gives you information on synchronous and asynchronous administration using administration commands. Chapter 12 includes a description of the administration commands, and of the functions that you can execute with these commands.

Chapter 13 contains information on administering local message queues and on the administration of printers via a printer control LTERM.

For technical reasons the printed manual is divided into two volumes.

Wherever the term Unix system is used in the following, then this should be understood to mean a Unix-based operating system such as Solaris or HP-UX.

Wherever the term Linux system is used in the following, then this should be understood to mean a Linux distribution such as SUSE or Red Hat.

Wherever the term Windows system or Windows platform is used below, this should be understood to mean all the variants of Windows under which openUTM runs.

1.2 Summary of contents of the openUTM documentation

This section provides an overview of the manuals in the openUTM suite and of the various related products.

1.2.1 openUTM documentation

The openUTM documentation consists of manuals, the online help systems for the graphical administration workstation openUTM WinAdmin and the graphical administration tool WebAdmin, and a release note for each platform on which openUTM is released.

Some manuals are valid for all platforms, and others apply specifically to BS2000 systems or to Unix, Linux and Windows systems.

All the manuals are available as PDF files on the internet at

http://manuals.ts.fujitsu.com

On this site, enter the search term “openUTM V6.4“ in the **Search by product** field to display all openUTM manuals of version 6.4.

The manuals are included on the Enterprise DVD with open platforms and are available on the WinAdmin DVD for BS2000 systems.

The following sections provide a task-oriented overview of the openUTM V6.4 documentation. You will find a complete list of documentation for openUTM in the chapter on related publications at the back of the manual.

Introduction and overview

The **Concepts and Functions** manual gives a coherent overview of the essential functions, features and areas of application of openUTM. It contains all the information required to plan a UTM operation and to design an UTM application. The manual explains what openUTM is, how it is used, and how it is integrated in the BS2000, Unix, Linux and Windows based platforms.

Programming

- You will require the **Programming Applications with KDCS for COBOL, C and C++** manual to create server applications via the KDCS interface. This manual describes the KDCS interface as used for COBOL, C and C++. This interface provides the basic functions of the universal transaction monitor, as well as the calls for distributed processing. The manual also describes interaction with databases.
- You will require the **Creating Applications with X/Open Interfaces** manual if you want to use the X/Open interface. This manual contains descriptions of the UTM-specific extensions to the X/Open program interfaces TX, CPI-C and XATMI as well as notes on configuring and operating UTM applications which use X/Open interfaces. In addition, you will require the X/Open-CAE specification for the corresponding X/Open interface.
- If you want to interchange data on the basis of XML, you will need the document entitled openUTM **XML for openUTM**. This describes the C and COBOL calls required to work with XML documents.
- For BS2000 systems there is supplementary documentation on the programming languages Assembler, Fortran, Pascal-XT and PL/1.

Configuration

The **Generating Applications** manual is available to you for defining configurations. This describes for both standalone UTM applications and UTM cluster applications how to use the UTM tool KDCDEF to

- define the configuration
- generate the KDCFILE
- and generate the UTM cluster files for UTM cluster applications

In addition, it also shows you how to transfer important administration and user data to a new KDCFILE using the KDCUPD tool. You do this, for example, when moving to a new openUTM version or after changes have been made to the configuration. In the case of UTM cluster applications, it also indicates how you you can use the KDCUPD tool to transfer this data to the new UTM cluster files.

Linking, starting and using UTM applications

In order to be able to use UTM applications, you will need the **Using openUTM Applications** manual for the relevant operating system (BS2000 or Unix, Linux and Windows systems). This describes how to link and start a UTM application program, how to sign on and off to and from a UTM application and how to replace application programs dynamically and in a structured manner. It also contains the UTM commands that are available to the terminal user. Additionally, those issues are described in detail that need to be considered when operating UTM cluster applications.

Administering applications and changing configurations dynamically

- The **Administering Applications** manual describes the program interface for administration and the UTM administration commands. It provides information on how to create your own administration programs for operating a standalone UTM application or a UTM cluster application and on the facilities for administering several different applications centrally. It also describes how to administer message queues and printers using the KDCS calls DADM and PADM.
- If you are using the graphical administration workstation **openUTM WinAdmin** or the Web application **openUTM WebAdmin**, which provides comparable functionality, then the following documentation is available to you:
  - A **description of WinAdmin** and **description of WebAdmin**, which provide a comprehensive overview of the functional scope and handling of WinAdmin/ WebAdmin. These documents are shipped with the associated software and are also available online as a PDF file.
  - The respective **online help systems**, which provide context-sensitive help information on all dialog boxes and associated parameters offered by the graphical user interface. In addition, it also tells you how to configure WinAdmin or WebAdmin in order to administer standalone UTM applications and UTM cluster applications.

For detailed information on the integration of openUTM WebAdmin in SE Server's SE Manager, see the SE Server manual **Operation and Administration**.

Testing and diagnosing errors

You will also require the **Messages, Debugging and Diagnostics** manuals (there are separate manuals for Unix, Linux and Windows systems and for BS2000 systems) to carry out the tasks mentioned above. These manuals describe how to debug a UTM application, the contents and evaluation of a UTM dump, the behavior in the event of an error, and the openUTM message system, and also lists all messages and return codes output by openUTM.

Creating openUTM clients

The following manuals are available to you if you want to create client applications for communication with UTM applications:

- The **openUTM-Client for the UPIC Carrier System** describes the creation and operation of client applications based on UPIC. In addition to the description of the CPI-C and XATMI interfaces, you will find information on how you can use the C++ classes to create programs quickly and easily.
- The **openUTM-Client for the OpenCPIC Carrier System** manual describes how to install and configure OpenCPIC and configure an OpenCPIC application. It describes how to install OpenCPIC and how to configure an OpenCPIC application. It indicates what needs to be taken into account when programming a CPI-C application and what restrictions apply compared with the X/Open CPI-C interface.
- The documentation for the **JUpic-Java classes** shipped with **BeanConnect** is supplied with the software. This documentation consists of Word and PDF files that describe its introduction and installation and of Java documentation with a description of the Java classes.
- The **BizXML2Cobol** manual describes how you can extend existing COBOL programs of a UTM application in such a way that they can be used as an XML-based standard Web service. How to work with the graphical user interface is described in the **online help system**.
- If you want to provide UTM services on the Web quickly and easily then you need the manual **WebServices for openUTM**. The manual describes how to use the software product WS4UTM (WebServices for openUTM) to make the services of UTM applications available as Web services. The use of the graphical user interface is described in the corresponding **online help system**.

Communicating with the IBM world

If you want to communicate with IBM transaction systems, then you will also require the manual **Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications**. This describes the CICS commands, IMS macros and UTM calls that are required to link UTM applications to CICS and IMS applications. The link capabilities are described using detailed configuration and generation examples. The manual also describes communication via openUTM-LU62 as well as its installation, generation and administration.

PCMX documentation

The communications program PCMX is supplied with openUTM on Unix, Linux and Windows systems. The functions of PCMX are described in the following documents:

- CMX manual “Betrieb und Administration“ (Unix-Systeme) for Unix, Linux and Windows systems (only available in German)
- PCMX online help system for Windows systems

1.2.2 Documentation for the openSEAS product environment

The **Concepts and Functions** manual briefly describes how openUTM is connected to the openSEAS product environment. The following sections indicate which openSEAS documentation is relevant to openUTM.

Integrating Java EE application servers and UTM applications

The BeanConnect adapter forms part of the openSEAS product suite. The BeanConnect adapter implements the connection between conventional transaction monitors and Java EE application servers and thus permits the efficient integration of legacy applications in Java applications.

- The manual **BeanConnect** describes the product BeanConnect, that provides a JCA 1.5- and JCA 1.6-compliant adapter which connects UTM applications with applications based on Java EE, e.g. the Oracle application server.
  The manuals for the Oracle application server can be obtained from Oracle.

Connecting to the web and application integration

You require the WebTransactions manuals to connect new and existing UTM applications to the Web using the product **WebTransactions**.

The manuals will also be supplemented by JavaDocs.

1.2.3 Readme files

Information on any functional changes and additions to the current product version described in this manual can be found in the product-specific Readme files.

Readme files are available to you online in addition to the product manuals under the various products at *http://manuals.ts.fujitsu.com*. For the BS2000 platform, you will also find the Readme files on the Softbook DVD.

Information on BS2000 systems

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

`SYSRME.<product>.<version>.<lang>`

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor. The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product's files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at *http://manuals.ts.fujitsu.com*.

Readme files on Unix and Linux systems

The Readme file and any other files, such as a manual supplement file, can be found in the *utmpath* under `/docs/`*language*.

Readme files on Windows systems

The Readme file and any other files, such as a manual supplement file, can be found in the *utmpath* under `\Docs\`*language*.

1.3 Innovations in openUTM V6.4

The following sections provide more detail on the innovations in the individual function areas.

1.3.1 New server functions

UTM cache in data spaces (BS2000 systems)

On BS2000 systems the UTM cache can be located on multiple data spaces. This provides benefits when a UTM application requires a very large cache.

The following interfaces have been changed to support this:

- Generation

  KDCDEF statement MAX... CACHE-SIZE=: New values PROGRAM-SPACE and DATA-SPACE.

- KDCADMI administration interface
  - Data structure *kc_max_par_str*: New field *cache_location* for the storage location of the UTM cache.
- KDCADM command interface
  - KDCINF SYSPARM: New output field CACHE-LOCATION for the storage location of the UTM cache.

Saving compressed data

UTM can compress data in secondary storage areas (GSSB, LSSB), long-term storages (TLS and ULS), and the communication area-program area, and thus reduce resource consumption. The UTM pages saved on average per data compression can be displayed using the administration functions.

The following interfaces have been changed to support this:

- Generation

  KDCDEF statement MAX: New operand DATA-COMPRESSION with which data compression is enabled or disabled.

- KDCADMI administration interface
  - Data structure *kc_curr_par_str*: New field *data_compression* for displaying and modifying the current compression setting and new field *avg_saved_pgs_by_compr* for the UTM pages saved on average per compression.

  - Data structure *kc_max_par_str*: New field *data_compression* for the generated compression setting.

- KDCADM command interface
  - KDCAPPL: New operand DATA-COMPRESSION for modifying the compression setting.
  - KDCINF STAT: Output of the pages saved on average per compression (AVG COMPRESS PAGES SAVED).
  - KDCINF SYSPARAM: Output of the generated compression setting (DATA-COMPRESSION (GEN)).

Page pool information

UTM outputs the current values for page pool utilization.

The administration interface was extended for this purpose:

- KDCADMI administration interface
  - New object type KC_PAGEPOOL for outputting the page pool information
  - New data structure *kc_pagepool_str* for the current utilization of the page pool.
- KDCADM command interface
  - KDCINF PAGEPOOL outputs the current utilization of the page pool.

Controlling the number of UTM system processes

The number of UTM system processes can be controlled by means of UTM generation, i.e. more or also fewer than the current maximum of three UTM system processes can be started.

The configuration interface was extended for this purpose:

- New KDCDEF operand MAX SYSTEM-TASKS. The default value *STD corresponds to the former setting.

IP subnets in Unix, Linux, and Windows systems

IP subnets can be defined for UTM applications in Unix, Linux, and Windows systems. IP subnets are used to enable communication partners access without name resolution using DNS or to assign a specific address range to LTERM pools.

The following interfaces have been changed to support this:

- Generation

  New KDCDEF statement SUBNET in order to define an IP subnet with the associated address ranges.

- KDCADMI administration interface
  - New object type KC_SUBNET in order to query information on IP subnets.
  - New data structure *kc_subnet_str* for the properties of an IP subnet.

Host name longer than 8 characters in Unix, Linux, and Windows systems

On Unix, Linux, and Windows systems, when a standalone or UTM cluster application is started, an attempt is also made to map a local host name to a UTM host name via the conversion file.

For all systems, the names of trace files for the ADMI trace and the traces for the X/Open interfaces CPI-C, XATMI, and TX have also been modified.

openUTM as 64-bit application on Windows systems

On Windows systems with a 64-bit operating system, openUTM is available as a 64-bit application.

KDCUPD thus also supports the transition from 32-bit to 64-bit architecture on Windows systems.

Dynamic XA connection on Windows systems

On Windows systems dynamic XA connection is possible as an alternative. The settings required for this must be made in the Windows Registry.

Authorization data for Oracle databases

For security reasons, the authorization data for Oracle databases can now only be specified in the UTM generation. Specification in the start parameters is no longer permitted, and is rejected with the message K237.

Multi-threaded network connection on Unix, Linux, and Windows systems

Only multi-threaded network connection is now supported on Unix, Linux, and Windows systems. The generation operand MAX NET-ACCESS is consequently no longer described.

Other changes

- Messages
  - New message K167 on Unix, Linux, and Windows systems after successful switching of stdout and stderr.
  - New message K199 when a task or process is terminated.
  - New message K237 when a user ID and/or a password was specified in clear text in the XA open string for an Oracle database.
  - Additional insert XPOSAS (OSI-TP ASSOCIATION REFERENCE) in some XAP-TP-messages.
- Administration commands
  - Output values of administration commands are output in floating point presentation if they require more space than is available.
  - KDCINF LTAC: new column D for LTACs which were deleted by means of dynamic administration.
- Starting a UTM(BS2000) application using the SDF command ENTER-PROCEDURE:
  - New start parameter ENTER-PROC-INPUT
  - New sample procedure START-APPL-ENTER-PROC
- Trace entries
  - When event exits (START, SHUT, etc.) are called, additional trace entries are written to the UTM-DIAGAREA.
- The PID is always output in its full length on Unix, Linux, and Windows systems.
- The TNS is not supported on Windows-64-bit systems. The address information must be specified in the UTM generation.

1.3.2 New client functions

There are no functional enhancements in V6.4.

1.3.3 New and modified functions for openUTM WinAdmin

WinAdmin supports all the new features of UTM V6.4 relating to the administration program interface. These include:

- The graphical display of the page pool utilization, displays for data compression, display of the cache location on BS2000 systems, and display of IP subnets on Unix, Linux, and Windows systems.
- Statistic collectors for the page pool utilization and the UTM pages saved by data compression.

1.3.4 New functions for openUTM WebAdmin

Support of new features in openUTM V6.4

WebAdmin supports all the new features of UTM V6.4 relating to the administration program interface. These include:

- The graphical display of the page pool utilization, displays for data compression, display of the cache location on BS2000 systems, and display of IP subnets on Unix, Linux, and Windows systems.
- Statistic collectors for the page pool utilization and the UTM pages saved by data compression.

Logging

In future it will be possible to log the sign-on and sign-off of a WebAdmin user and modifications to the WebAdmin configuration.

Enhanced integration into the SE Server

- The WebAdmin add-on supports audit logging of the SE Manager, i.e. specific modifications of the WebAdmin configuration are logged in the SE Manager logging.

  The logging data is also available for standalone WebAdmin.

- The current status of the WebAdmin add-on is displayed in the SE Manager and can be modified, i.e.:
  - whether the add-on is running is displayed

– Depending on the current status, the add-on can either be started, terminated, or terminated and restarted.

These two functions are available in the SE Manager only in M2000 V6.2A and higher.

Adaptation to WinAdmin

WebAdmin offers further additional functions which were previously only available in WinAdmin

- Command mode for direct entry of administration commands or normal UTM TACs.
- Enhanced functionality for statistic collectors:
  – The values of any number of statistic collectors can be saved, deleted, or output in table form simultaneously and copied from there.
  – The properties *Automatic Wake-Up*, *Sample Interval*, and *Statistic Values Lifetime* can be modified simultaneously for any number of statistic collectors.

1.4 Notational conventions

Metasyntax

The table below lists the metasyntax and notational conventions used throughout this manual:

Representation	Meaning	Example
UPPERCASE LETTERS	Uppercase letters denote constants (names of calls, statements, field names, commands and operands etc.) that are to be entered in this format.	`LOAD-MODE=STARTUP`
lowercase letters	In syntax diagrams and operand descriptions, lowercase letters are used to denote place-holders for the operand values.	`KDCFILE=filebase`
lowercase letters in italics	In running text, variables and the names of data structures and fields are indicated by lowercase letters in italics.	*utm-installationpath* is the UTM installation directory
`Typewriter font`	Typewriter font (Courier) is used in running text to identify commands, file names, messages and examples that must be entered in exactly this form or which always have exactly this name or form.	The call `tpcall`
{ } and |	Curly brackets contain alternative entries, of which you must choose one. The individual alternatives are separated within the curly brackets by pipe characters.	`STATUS={ ON | OFF }`
[ ]	Square brackets contain optional entries that can also be omiited.	`KDCFILE=( filebase [, { SINGLE| DOUBLE} ] )`
( )	Where a list of parameters can be specified for an operand, the individual parameters are to be listed in parentheses and separated by commas. If only one parameter is actually specified, you can omit the parentheses.	`KEYS=(key1,key2,...key`*n*`)`
Underscoring	Underscoring denotes the default value.	`CONNECT= { A/YES | NO }`
abbreviated form	The standard abbreviated form of statements, operands and operand values is emphasized in boldface type. The abbreviated form can be entered in place of the full designation.	**T**`RANSPORT-`**SEL**`ECTOR=c'C'`

Representation	Meaning	Example

. . .	An ellipsis indicates that a syntactical unit can be repeated.
	It can also be used to indicate sections of a program or syntax description etc.
		`Start KDCDEF`
		`:`
		`:`
		`OPTION`
		`DATA=statement_file`
		`:`
		`END`

Other symbols

B
B
This symbol is used in the left-hand margin to indicate BS2000 system specific elements of a description.

X
X
This symbol is used in the left-hand margin to indicate Unix and Linux system specific elements of a description.

W
W
This symbol is used in the left-hand margin to indicate Windows specific elements of a description.

B/X
B/X
This symbol is used in the left-hand margin to indicate parts of the description that are only relevant for openUTM on BS2000, Unix and Linux systems.

B/W
B/W
This symbol is used in the left-hand margin to indicate parts of the description that are only relevant for openUTM on BS2000 and Windows systems.

X/W
X/W
This symbol is used in the left-hand margin to indicate parts of the description that are only relevant for openUTM on Unix, Linux and Windows systems.

Indicates references to comprehensive, detailed information on the relevant topic.

Indicates notes that are of particular importance.

Indicates warnings.

X/W
X/W
X/W
utmpath
On Unix, Linux and Windows systems, designates the directory under which openUTM was installed.

B
B
$userid
On BS2000 systems, designates the user ID under which openUTM was installed.

2 Overview of openUTM administration

The term “administration” covers all activities involved in the control and administration of the current application. “Administering” means adapting the application to changing circumstances and requirements without interrupting the application run.

To help you administer your UTM application, openUTM provides you with the interfaces and tools in the following list.

- The command interface on which the basic administration functions are available. This is implemented in the KDCADM administration program.
- The KDCADMI program interface for administration which you can use to generate administration programs specifically tailored to your application. The UTM administration functions are provided at this program interface.
- The PADM and DADM calls at the KDCS program interface with which you administer local message queues and printers, enabling you to control the output of print jobs. The UTM program units KDCDADM and KDCPADM provide you with all the functions of the KDCS calls DADM and PADM (see the section “UTM program units for DADM and PADM functions” on page 820).
- The openUTM component WinAdmin with which you can administer several UTM applications in a network from the graphical user interface on your PC.
- The openUTM WebAdmin component that provides a Web application for the administration of UTM applications.

B WebAdmin can be integrated into the SE Manager as an add-on.

B B B
- The administration tool CALLUTM with which you can start also administration services in UTM applications while in a BS2000 task, and which enables you to call up administration commands.

B B B B
- The KDCISAT and KDCMSAT commands (dialog transaction codes) with which you can control the SAT logging function for your application. These commands are described in the openUTM manual “Using openUTM Applications on BS2000 Systems”.

openUTM provides you with a comprehensive range of administration functions via the command interface and the program interfaces KDCADMI and KDCS, enabling you to obtain optimum performance and flexibility from your application, and ensuring that the application can operate without interruption (7∗24-hour operation). You can, for example, perform the following actions:

- Check the performance of the application by querying information about the current utilization level of the application, diagnosing performance bottlenecks and errors and, where necessary, taking measures to improve performance.
- Replace parts of the application program or the entire application program at runtime. This enables you to modify program units during the application run or to add new program units.
- Assign the restart information and/or print queues on terminals and printers where hardware faults arise to other terminals or printers. This enables the user to continue work from a different terminal, or to redirect print jobs to an intact printer.
- Disable/enable clients, printers, LTERM pools, user IDs, services and the connection points for communication partners (LTERM, LPAP and OSI-LPAP partners) where necessary.
- Establish and shut down connections to clients, printers and partner applications or switch to replacement connections.
- Request information about the configuration of an application and the current settings for application and operating parameters.
- Modify the configuration of an application at runtime by adding to the configuration services, user IDs, clients, printers, connections and session names for distributed processing by means of LU6.1, key sets and transaction codes for partner applications or by deleting them from it.
- Administer TAC, USER and temporary queues as well as the local message queues of LTERM partners and transaction codes.
- Terminate an application.

You can call up the administration functions of openUTM (with the exception of the SAT administration command) in dialog mode or by means of message queuing. The message queuing form of administration for a UTM application involves the use of “programmed administrators”, i.e. you can generate programs which execute administration functions at a given time (DPUT call) or in response to specific events. The program interface calls and administration commands can, in particular, be called by the MSGTAC event service.

You can also take advantage of the opportunities offered by the user-specific message destinations. These message destinations allow you to read messages in a TAC or USER queue, for example, by means of the KDCS program interface and the DGET function. With this function and corresponding follow-up processing, you can design MSGTAC-like programs that respond specifically to a message.

For information on automatic administration refer to chapter “Automatic administration” on page 147.

UTM administration privileges are required for all administration functions which involve write access to configuration data of the application. There is also a slightly lower level of authorization which entitles users to use administration functions which have read-only access to the application data.

For details of the authorizations concept, see chapter “Access rights and data access control” on page 153ff.

The following section provides a summary of the range of functions for individual interfaces and tools and also describes the differences between them and their respective areas of application.

2.1 Command interface

openUTM is supplied with the standard administration program KDCADM in which some of the functions at the program interface for administration (KDCADMI) are implemented. The command interface for administration supports some of the functions of the program interface for administration (KDCADMI).

KDCADM provides the basic administration functions which you need in order to ensure that the application is available continuously, and to check the performance of the application. KDCADM is not able to add new objects dynamically or to delete objects from the configuration.

In order to call up individual KDCADM functions, you must assign specified transaction codes to the program KDCADM. These transaction codes are referred to as administration codes.

There is a dialog transaction code (dialog command) for each KDCADM function and an asynchronous transaction code (asynchronous command). You can therefore call the KDCADM administration functions synchronously in dialog mode or asynchronously by means of message queuing.

When you call a command you can specify operands. With these operands, you can define the type of action which is to be executed and specify the objects in the application to which the action must relate. The operands are identical for the respective dialog and asynchronous commands.

The KDCADM administration commands and their operands are described in chapter “Administration commands - KDCADM” on page 663ff.

Administration commands can only be entered in line mode. Similarly, administration commands are also output in line mode. It is not possible to use formats.
For this reason, KDCINF, for example, only returns the most important data in response to a query about object properties and application parameters.

You will find information about the layout of output for administration in message queuing mode in chapter “Administration using commands” on page 109ff.

You will need to use KDCDEF to generate both the administration commands you wish to use at runtime and the administration program KDCADM. Alternatively, you can use the KDCADMI program interface to include them dynamically. You must always enter the KDCSHUT command in the configuration for your application. You can use KDCSHUT to terminate the application normally.

The following table contains a summary of KDCADM functions and the commands which you use to call up these functions.

	KDCADM administration function	Dialog command	Asynchronous command
B B	Adjust the settings for application parameters and timers, define current number of processes for the application, establish connections to the printers for which print jobs exist, replace the entire application program, replace sections of the application marked in the common memory pool for replacement.	KDCAPPL	KDCAPPLA
	Producing diagnostic documentation, e.g. request a UTM diagnosis dump	KDCDIAG	KDCDIAGA
	Exchange master LTERMs of two LTERM bundles	KDCBNDL	KDCBNDLA
	Query properties of objects and the current settings of application parameters, request statistical information	KDCINF	KDCINFA
	Switch the user log file to the next file generation	KDCLOG	KDCLOGA
	Disable/enable LTERM partners, set up and shut down connections	KDCLTERM	KDCLTRMA
	Change the number of clients approved for an LTERM pool	KDCPOOL	KDCPOOLA
	Replace load modules/shared objects/DLLs in the application	KDCPROG	KDCPROGA
	Disable/enable clients/printers, set up and shut down connections	KDCPTERM	KDCPTRMA
	Terminate the UTM application run	KDCSHUT	KDCSHUTA
	Switch the system log file (SYSLOG) of the application, activate/deactivate size monitoring, modify the control value for size monitoring, query information via the SYSLOG	KDCSLOG	KDCSLOGA
	Change the assignment of clients/printers to LTERM partners	KDCSWTCH	KDCSWCHA
	Disable/enable transaction codes (local services)	KDCTAC	KDCTACA
	Modify the maximum number of processes entitled to process jobs for a TAC class simultaneously	KDCTCL	KDCTCLA
	Disable/enable user IDs, change passwords	KDCUSER	KDCUSERA
B B	Disable/enable multiplex connections, set up and shut down connections	KDCMUX	KDCMUXA
B	Send a message to one or more dialog terminals	KDCSEND	KDCSENDA
	The following functions are available for the administration of server-server communication via LU6.1 and OSI TP:		
	Set up and shut down logical connections to partner applications, switch replacement connections to OSI TP partners, disable/enable LPAP or OSI-LPAP partners, change timers for monitoring sessions and associations.	KDCLPAP	KDCLPAPA
	Set up and shut down logical connections for a session	KDCLSES	KDCLSESA

KDCADM functions and transaction codes

KDCADM administration function	Dialog command	Asynchronous command
Disable/enable a remote service (LTAC) for the local application, and adjust timer settings for monitoring the establishment of sessions/associations and their response times.	KDCLTAC	KDCLTACA

KDCADM functions and transaction codes

2.2 KDCADMI program interface

You can use the program interface for administration (KDCADMI) to create administration programs specifically tailored to suit your application. This program interface is provided in C/C++ and COBOL. This manual describes the program interface for C/C++. Since the COBOL interface is broadly similar to the C/C++ interface, you can also use the description in this manual as a guide when creating COBOL administration programs. For additional information about creating administration programs in COBOL, see also the appendix, starting on page 839.

The program interface offers functions which go beyond the basic administration functions of KDCADM. The KDCADMI program interface also offers you the following additional functions:

- Functions with which you can modify the configuration dynamically:
  You can add new services (program units, transaction codes), clients, printers, user IDs, connections and session names for distributed processing by means of LU6.1, key sets, transaction codes for partner applications and service-controlled queues to the configuration dynamically, delete them from the configuration or change the properties of objects or application parameters.
- Inverse KDCDEF:
  You can generate control statements for generation tool KDCDEF from the configuration information stored in the KDCFILE.
  This means that changes to the configuration made during the application run can be transferred when the application is regenerated.
- Output all configuration data when information is requested:
  When information is requested for individual objects or application parameters, all the configuration data stored in the KDCFILE for this object or parameter is returned. In a custom-made administration program you can analyze and process exactly the data that is of interest for a given application. When requesting information, you can restrict output to those objects which satisfy particular criteria by entering these selection criteria when you make the call.

The following table lists the functions of KDCADMI and the operation codes which are used to call up program functions.

The KDCADMI program interface and all data structures are described in chapter “Program interface for administration - KDCADMI” on page 159ff.
Information about dynamic administration and inverse KDCDEF can be found in chapter “Changing the configuration dynamically” on page 67 and chapter “Generating KDCDEF statements from the KDCFILE” on page 101.

	KDCADMI Function	KDCADMI operation code
B B X/W X/W X/W	Replace the entire application program without shutting down the application. Replace sections of the application in the common memory pool which are marked for replacement. When doing this, you must specify whether the next higher version, the next lower version or the current version of the application program is to be loaded.	KC_CHANGE_APPLICATION
	Generate a UTM diagnosis dump without terminating the application.	KC_CREATE_DUMP
	Extend the configuration of an application dynamically to include new services (program units, transaction codes), clients, printers, user IDs, connections and session names for distributed processing by means of LU6.1, key sets, transaction codes for partner applications and service-controlled queues.	KC_CREATE_OBJECT
	Start an inverse KDCDEF run online	KC_CREATE_STATEMENTS
	Delete clients, printers, user IDs, services, connections and session names for distributed processing by means of LU6.1, key sets, transaction codes for partner applications and service-controlled queues from the configuration of the application.	KC_DELETE_OBJECT
	Generate, activate or delete RSA key pair. Read public key of RSA key pair.	KC_ENCRYPT
	Query the names and properties of objects, the current settings of application parameters and statistical information	KC_GET_OBJECT
	Permit a new sign-on for all users or for an individual user still recorded as signed on at a failed node application or who have/has a service bound to the failed node application,. Release cluster user file lock after incorrectly terminated KDCDEF run. (Only in UTM cluster applications)	KC_LOCK_MGMT

Administration functions in the program interface for administration

KDCADMI Function	KDCADMI operation code
Modify the properties of objects or application parameters, e.g.: change the settings for application parameters and timers, define current process numbers for the application, activate/deactivate traces, replace load modules/shared objects/DLLs in the application, disable/enable user IDs, transaction codes, clients/printers or connections to partner applications, establish and shut down connections to clients, printers and partner applications, activate OSI TP replacement connections, change the number of clients approved for an LTERM pool, change the assignment of clients/printers to LTERM partners, reset counter for statistics data, change keys in key sets, change the data access control for transaction codes, users and TAC queues.	KC_MODIFY_OBJECT
Import application data from a terminated into a running node application (only for UTM cluster applications).	KC_ONLINE_IMPORT
Roll back transaction in PTC state (prepare to commit).	KC_PTC_TA
Send message to a dialog terminal or to all active dialog terminals.	KC_SEND_MESSAGE
Terminate the UTM application run.	KC_SHUTDOWN
Establish connections to printers for which print jobs exist.	KC_SPOOLOUT
Switch the system log file (SYSLOG) in the application, activate/deactivate size monitoring on/off, modify the control value for size monitoring, request information via SYSLOG	KC_SYSLOG
Determine IP addresses of generated communication partners: on BS2000 systems: only for T-PROT=SOCKET	KC_UPDATE_IPADDR
Switch the user log file(s) to the next generation of file	KC_USLOG

B

Administration functions in the program interface for administration

In addition to the greater range of functions that you can use in administration programs you write yourself, administration programs which utilize the functions of the program interface also offer the following advantages:

- For administration by means of message queuing, you can choose any recipient for the results. This means that, depending on the result of a KDCADMI call, you can call up various follow-up transactions.

  This yields advantages for automatic and programmed administration.

- The results of an administration call can be analyzed and further processed in the program unit containing the The number of administration calls which are subject to transaction management and which are to be executed in a single transaction is, however, limited by the generated size of the restart area (generation statement MAX, parameter RECBUF, see openUTM manual “Generating Applications”).

B

- You can use formats for the entry and output of administration programs.

Calls for administration functions must be made between the KDCS calls INIT and PEND. The data structures required for the exchange of data between openUTM and the program are predefined. For C/C++, the data structures are provided in the include file *kcadminc.h* (Unix, Linux and Windows systems) or in the include member *kcadminc.h* in the SYSLIB.UTM.064.C library (BS2000 systems).

For information about setting up a program, see chapter chapter “Writing your own administration programs” on page 115.

openUTM on BS2000, Unix, Linux and Windows systems use the identical data structures. These data structures contain a few fields which only relate to one of these operating systems. In the other operating system, binary zeroes must be entered in these fields. The program is able to determine which operating system it is running on with the aid of a KDCADMI call.

Since the KDCADMI calls and the data structures used are platform-independent, you can use KDCADMI to create administration programs which:

- allow the user to administer several UTM applications from one “central” location. These UTM applications can even be running on different platforms. In particular, you can administer UTM applications on BS2000 systems from a UTM application on Unix, Linux or Windows system and vice versa. These applications can be running under different versions of openUTM.
- are portable. You can compile the same source of an administration program on any of the three platforms and link it to a UTM application from there.

For information on central administration of applications, see chapter “Central administration of several applications” on page 121ff.

KDCADMI calls can, with one exception (termination of application run: KC_SHUTDOWN with subcode KC_KILL), be submitted in dialog as well as asynchronous services.

These dialog services can be started by users at the terminal, via UPIC clients or OpenCPIC partners, or by a partner application.
The asynchronous services can be started by users at the terminal, by partner applications and by OpenCPIC partners or from a program unit.

The program interface for administration is subject to the compatibility guarantee, i.e. it is offered in source-compatible across several different versions of openUTM. For this reason, administration programs do not need to be adapted to changes of version: instead they should be recompiled as they are and then linked into a UTM application running under the follow-up version.

2.3 Sample programs

openUTM is shipped with sample programs in the form of source code and object modules. You can use these as a basis for your own administration programs, modify them as required, compile them and integrate them in your application. The sample programs are the programs HNDLUSR (only BS2000 systems), ENCRADM, SUSRMAX and COBUSER. You will find an introduction to these in the section “Sample programs” on page 844).

2.4 PADM, DADM for administering message queues and printers

You can use the PADM and DADM calls at the KDCS program interface to administer the message queues and printers for an application and to control the printer output.

For example, you can change the sequence of the jobs or messages in a queue, delete jobs or messages from the queues, generate printer pools and, in the event of a printer fails, you can redirect print jobs to another printer. In addition, you can move messages from the dead letter queue into other message queues in order to edit them.

The calls PADM and DADM enable users or clients with no administration privileges to administer printers, control printer output and administer the message queues for a printer. In other words, “normal” users can administer their own “local” printers and administer the print jobs sent to these printers. Administration can be performed from the print control LTERM to which the printer being administered is assigned.

PADM and DADM can also be used by the event service MSGTAC. The MSGTAC routine can be started automatically if a printer fails and appropriate action can be taken in response to PADM and DADM calls.

Program units KDCDADM and KDCPADM are supplied with openUTM. These sample programs provide access to all services requested by the DADM and PADM calls without requiring you to create your own program units.

The PADM and DADM calls and the KDCDADM and KDCPADM programs are described in chapter “Administering message queues and controlling printers” on page 799ff.

W W W

Print output from a UTM application is not supported by openUTM on Windows systems. Consequently, the PADM function in UTM applications on Windows systems is not relevant.

2.5 Administration tool CALLUTM

B B B B B B

CALLUTM is an UPIC client on a BS2000 system with the aid of which you can call UTM services from any BS2000 task. Using CALLUTM’s SDF interface, you can start administration services in UTM applications on the same computer and also on other computers on the network. In particular, you can administer several UTM applications in the network centrally. These can either be UTM applications on BS2000 systems or UTM on Unix, Linux or Windows systems. CALLUTM is capable of running in dialog or in batch mode.

B

CALLUTM is described in the appendix, starting on page 847.

2.6 openUTM WinAdmin and openUTM WebAdmin

The openUTM components WinAdmin and WebAdmin provide you with a convenient graphical user interface for the administration of individual or multiple UTM applications.

WinAdmin and WebAdmin both provide much the same function scope.

The UTM applications may be distributed across the network. They can run on all approved platforms and possess different version levels. Both WinAdmin and WebAdmin support the full function scope of the program interface offered by the version in question.

The UTM applications requiring administration can be grouped into pools which can then be administered jointly. For example, it is possible in **one** step to modify objects in **several** applications.

You have to generate the KDCWADMI administration program and the relevant transaction code KDCWADMI, in order to be able to administer a UTM application through WinAdmin or WebAdmin. Specify ADMIN=YES for the transaction code. KDCWADMI is part of the delivery scope of openUTM.

You can also use WinAdmin and WebAdmin to start and end UTM applications. When you start a UTM application, the system assumes that openFT is available on the relevant computer. Consequently the openUTM WebAdmin add-on in the SE Manager cannot start any UTM applications.

Security

The full range of UTM security functions, starting with access control using UTM user IDs and passwords through to password and data encryption, is at your disposal in WinAdmin and WebAdmin.

WinAdmin and WebAdmin, moreover, also offer their own user concept, allowing you to define several users with different rights, from read-only users through to “master” users, i.e. the WinAdmin or WebAdmin administrators. Each user’s access to WinAdmin or WebAdmin is password-protected.

Differences between WinAdmin and WebAdmin

While openUTM WinAdmin is a Java application that runs on Windows, Unix and Linux systems, openUTM WebAdmin is a web application which can be accessed from any computers or mobile devices using a web browser.

Using WinAdmin it is possible to modify objects in multiple applications in a single step or to combine multiple administration steps in a single transaction.

You will find an introduction to WinAdmin and WebAdmin in section “Administration using WinAdmin and WebAdmin” on page 123.

3 Administering objects and setting parameters

This chapter provides a summary of the options made available by UTM's administration functionality. A few application areas of UTM administration are illustrated here by way of example. The administration commands and program interface calls with which you can perform individual actions are merely referred to.

The chapter "Program interface for administration - KDCADMI" on page 159ff and the chapter "Administration commands - KDCADM" on page 663ff contain a detailed description of the actions which you are able to perform with the aid of the program interface and the administration commands.

The present chapter does not provide details of the administration functions for dynamically entering new objects in the configuration, changing object properties or deleting objects. These administration functions are described in chapter "Changing the configuration dynamically" on page 67.

The following symbols are used in the ensuing description:

refers to the administration command with which you can perform actions. Only the dialog command is indicated in each case. However, you can also use the appropriate asynchronous command to execute the specified actions (see table on page 35).

refers to the function call at the program interface for administration with which you can execute the required administration function.

You call also use all of the functions described in this section with the administration tools, WinAdmin and WebAdmin.

3.1 Information functions in openUTM

openUTM provides you with information functions with which you can obtain an overview of the configuration of your application, the settings for application parameters and the current utilization level of the application. You can call the information functions of UTM administration with:

KDCINF

KC_GET_OBJECT

These information functions can also be utilized by users who do not have administration privileges (see chapter “Access rights and data access control” on page 153ff).

With the aid of information functions, you can, for instance, arrange for output of the following information:

- Application and system parameters defined during KDCDEF generation with the MAX statement
  (page 733 / page 624).
- Number of processes currently active for the application, maximum number of processes that can be available for asynchronous processing at one time, maximum number of processes that are available for processing services at one time and that contain blocking calls, such as the KDCS call PGWT or the XATMI call tpcall
  (page 733 / page 652).
- Data about the current utilization level of the application. This information can, for example, include utilization of the page pool or the cluster page pool, the total number of messages being exchanged, the number of users and clients signed on, the number of services open at the present time, the number of transactions performed per unit of time, the number of jobs buffered in the message queues etc.
  (see page 726 and 733 / page 600).
- Current settings for the timers. In UTM, for example, timers are defined for assigning and waiting for resources, waiting for an answer from a dialog partner both during and outside of a transaction, waiting for confirmations, and waiting for a connection or session to be established (see page 733 / page 656).
- Configuration data on all objects which appear in the configuration. This includes the names and logical properties defined when adding objects to the configuration. It also includes control values for the message queues, the number of LTERM partners in an LTERM pool or the maximum number of parallel connections generated to an OSI TP partner application.

- Status of individual communication partners and printers in the application, and of connections to these. For example, the output can show whether the communication partner is connected to the application and the length of time that such a connection has been in existence, as well as whether or not the communication partner is currently disabled, the number of messages exchanged on the connection, and whether automatic connection setup is generated.
- Maximum number of objects of a given type that the configuration of the application can maintain.
- Number of objects that can still be added dynamically to the configuration.

Details of which specific data are returned is described in section “Data structures used to pass information” on page 443 for queries with KC_GET_OBJECT and as of page 703 for queries using the administration command KDCINF.

With information queries you can specify the selection criteria, i.e. you can request information on objects which have particular properties, e.g.:

- all LU6.1 connections currently established
- the association ID of all associations currently established to an OSI TP partner application
- all clients and printers currently connected to the application
- all users currently connected to the application
- all LTERMs of a connection bundle or all (OSI-)LPAPs of a LPAP bundle

3.2 Performance check

openUTM offers you numerous functions which you can use to obtain up-to-date information about the utilization level of the application, to diagnose bottlenecks and to initiate actions to improve performance.

Reasons for performance bottlenecks can include such things as:

- Increased requirements on service calls during peak times
- Too many users/clients are working with the application at the same time
- The processes that are available to the application are occupied by jobs for an extended period because they have to wait for resources locked by other processes
- Processing of a large number of asynchronous jobs impairs dialog operation
- Too many long-running program units are running at the same time, e.g. program units which conduct a search of all data for specific information
- Many program units containing blocking calls are running at the same time, e.g. the KDCS call PGWT or the XATMI call *tpcall*. During the waiting period, each of these program units occupies a process in the application on an exclusive basis.
- With distributed processing using OSI TP or LU6.1, the system waits long for an association or session to be assigned
- Frequent I/O accesses to the page pool
  Frequent read accesses may indicate that the cache generated for the UTM application is too small.
- Bottlenecks to connections to communication partners in the application

3.2.1 Information about the utilization level of the application

On the basis of data relating to the current and maximum utilization level of the application and of individual objects supplied by the information functions of UTM, you can identify pending bottlenecks and introduce measures in good time to prevent these bottlenecks from occurring.

The value for the maximum utilization of the UTM application takes in to account the period from the time of the startup of the application or from the last time the value was reset to the time of the query.

The value for the current utilization of the application provides information on the last closed 100 second interval before the time of the query.

You can obtain important data for performance control purposes with the following calls:

KDCINF STATISTICS or SYSPARM (general data)
KDCINF object type (query about data for individual objects)
The data actually returned by KDCINF STATISTICS are described from page 726.

KC_GET_OBJECT with *obj_type*=KC_CURR_PAR (general data)
For queries about object-related data, enter the type of the object in *obj_type*.
The data actually returned in response to queries with KC_CURR_PAR is described from page 600. Object-specific data can be found in section “Data structures for describing object properties” on page 445.

KC_GET_OBJECT with *obj_type*=KC_CLUSTER_CURR_PAR
Supplies information about the occupancy of the cluster page pool in UTM cluster applications, see page 592.

If the information functions mentioned above indicate bottlenecks, you should carry out a more detailed analysis using the UTM metering monitor KDCMON which gathers statistical data, e.g. on the utilization level of the application, the progress of application program units, and the time needed to process a job. With the aid of system administration, you can activate KDCMON and deactivate it again after a desired period of time while the system is running. You can evaluate the data thus obtained using the UTM tool KDCEVAL.

KDCAPPL KDCMON

KC_MODIFY_OBJECT with *obj_type*=KC_DIAG_AND_ACCOUNT_PAR

KDCMON and the tool KDCEVAL are described in the openUTM manual “Using openUTM Applications”, where you will also find interpretation aids for the statistics produced by KDCMON and the measures you can take to eliminate bottlenecks.

For performance control purposes, you also have the software monitor openSM2. openSM2 supplies statistical data on the performance of the complete application program and the utilization level of the system resources. You can activate/deactivate the supply of data to openSM2 through Administration. For further information on openSM2 also refer to the openUTM manual “Using openUTM Applications”.

KDCAPPL SM2

KC_MODIFY_OBJECT with *obj_type*=KC_MAX_PAR

3.2.2 Diagnosing errors and bottlenecks

openUTM provides the following functions which assist you during the diagnosis of performance bottlenecks and incorrect program behavior:

- You can check the maximum utilization of an application in a particular period.
- You can log events in the form of UTM messages in the SYSLOG.
- In order to diagnose bottlenecks and errors in connections to communication partners, you can activate the UTM BCAM trace or the OSS trace. The UTM BCAM trace can be activated for all connections, for a specific user only or just for connections to specific partner applications and clients.
- You can enable the CPI-C trace, TX trace or XATMI trace to diagnose errors that occur in program units that use the X/Open interfaces CPI-C, TX or XATMI.
- You can enable the ADMI trace to diagnose errors that occur at the administration program interface (KDCADMI).
- You can activate test mode. Test mode is used to generate diagnostic documentation when errors occur in the UTM system code. Since test mode has a negative impact on UTM application performance, you should only activate test mode when requested to do so by Systems Support. In test mode, additional internal UTM plausibility checks are conducted and internal trace information is logged.
- You can request a diagnostic dump without having to interrupt the execution of the application. In this case, you can do the following by issuing a command or via the program interface:
  - immediately request a general diagnosis dump. This has the ID DIAGDP ID.
  - or request a dump as soon as a particular event (message, KDCS return code, sign-on return code) is generated by openUTM. The dump ID is dependent on the event. You must first activate test mode since the dump is only written when test mode is active.

KDCDIAG

KC_MODIFY_OBJECT with *obj_type*=KC_DIAG_AND_ACCOUNT_PAR

3.2.3 Possible measures

The following section describes some of the measures you can take to avoid performance bottlenecks or to remedy existing bottlenecks.

Increasing the total number of processes for an application

If extended wait periods arise when processing jobs, particularly in dialog mode, you can increase the number of processes in which the application program runs.

This makes particular sense in the event that the current application load rises above 80 % and at the same time sufficient system resources are still free (memory space, CPU capacity). This value should fall again after the total number of processes has been increased sufficiently.

The maximum permitted number of processes is defined in MAX TASKS during KDCDEF generation.This maximum number cannot be increased at the administrative level. However, if the number of processes currently set is less than this maximum number, you can start additional processes for the application.

KDCINF SYSPARM:
Query the current maximum number of processes and the maximum permitted number of processes.
KDCAPPL TASKS: define a new number of processes.

KC_GET_OBJECT with *obj_type*=KC_TASKS_PAR:
Query the maximum permitted number of processes and the current number of processes.
KC_MODIFY_OBJECT with *obj_type*=KC_TASKS_PAR: change the number of processes.

Reducing the total number of processes for an application

Because of the possibility of load fluctuations, it is generally not sensible to reduce the total number of processes if the application is not loaded to capacity part of the time.

The total number of processes should only be reduced when the computer as a whole encounters a bottleneck which leads to reduced throughput and/or slower response times on the part of the application.

If you reduce the total number of processes, you must note the following points:

- If the total number of processes is reduced to such a level that it is less than the currently set maximum number of processes that can be used at the same time for asynchronous processing (hereafter referred to as ASYNTASKS), openUTM resets the value for ASYNTASKS to the specified total number of processes. For subsequent changes to the total number of processes, openUTM adapts the value of ASYNTASKS automatically until the value is reached which was previously set by administration or in the startup parameter for ASYNTASKS.

  The same applies to the maximum number of program units with blocking calls (TASKS-IN-PGWT) permitted to run simultaneously. Note that the maximum number of processes must be at least 2 if a transaction code or a TAC class is generated with PGWT=YES or if the application is a UTM cluster application.

- If, in a dialog TAC class, the value for TASKS-FREE is greater than the current total number of processes, one process then continues to process the jobs going to this TAC class.

- If, in the application, job processing is priority controlled (TAC-PRIORITIES is generated), and the value for FREE-DIAL-TASKS is greater than the current total number of processes, one process then continues to process the jobs going to this TAC class.

To ensure that, after the total number of processes has been reduced, dialog operation is not impaired by long-running asynchronous services or by programs with blocking calls, it is advisable to adapt the value of ASYNTASKS and TASKS-IN-PGWT to reflect the reduction you make in the total number of processes, i.e. you should also reduce this value.

Reducing the number of processes available for asynchronous processing and for the processing of program units with blocking calls

If the dialog mode for an application is delayed by time-consuming asynchronous processing (in other words, if dialog jobs wait because too many processes are handling asynchronous jobs at the same time), you can reduce the maximum number of processes (ASYNTASKS) that can be used at one time for asynchronous processing. This means that there remain more processes free for synchronous processing. The number of processes in ASYNTASKS is restricted by the maximum value generated in MAX ASYNTASKS.

You can occasionally set ASYNTASKS to 0. However, when doing so, you should note that all asynchronous jobs are placed in buffer storage in the page pool. If the page pool is not large enough, this can cause bottlenecks in the page pool.

When you reduce ASYNTASKS and if jobs are controlled through process restrictions for the individual TAC classes in your application (TAC-PRIORITIES is not generated), you must also note the following:

If an asynchronous TAC class exists for which the current value set in TASKS-FREE is greater than or equal to ASYNTASKS, then this TAC class is disabled, i.e. no further jobs are processed for this TAC class. In this instance, TASKS-FREE is the minimum number of processes which should be kept free for processing other jobs going to other asynchronous TAC classes.

To maintain a check, you should request information about the TAC classes after reducing the ASYNTASKS.

The same applies to the maximum number of processes (TASKS-IN-PGWT) in which program units with blocking calls are allowed to run at the same time. In contrast to ASYNSTASKS, however, note that you *cannot* set the value to 0, if such tasks exist.

KDCINF SYSPARM:
Display current settings
KDCAPPL ASYNTASKS / TASKS-IN-PGWT: change number of processes

KC_GET_OBJECT with *obj_type*=KC_TASKS_PAR:
Determine generated maximum number and currently set number of processes
KC_MODIFY_OBJECT with *obj_type*=KC_TASKS_PAR:
change number of processes

In applications without TAC-PRIORTIES:
changing the number of processes for individual TAC classes

If your application is generated with TAC classes, you can define a specific maximum number of processes for each TAC class, i.e. the number of processes able to process jobs in one TAC class, and you can change this number if so required.

When creating the transaction code, you indicate the TAC class to which a transaction code is to belong. You can therefore group transaction codes belonging to long-running program units into one TAC class or several TAC classes. The proportion of processes in the application that are authorized to process jobs in this TAC class at the same time can then be set by you at a level which reflects the utilization of that application. In the case of dialog TAC classes, at least one process must be allowed to process jobs in the TAC class. In the case of asynchronous TAC classes, the number can be reduced to 0.

In particular you should group the dialog TACs in program units containing blocking calls (e.g. KDCS call PGWT, or XATMI call *tpcall*) in one TAC class (with PGWT=YES). After a blocking call, the program unit waits until the data required for continuing the program has been received. For this period of time, the program unit and the related transaction code assigns a process in the application on an exclusive basis. If several similar program units are running concurrently, this can cause other jobs to remain waiting in the queue because no processes are available to process them. The performance of the application is thus severely impaired. The wait time following a blocking call can also be restricted using the timer PGWTTIME (see below).

KDCINF TACCLASS:
Determine current setting
KDCTCL: change number of processes

KC_GET_OBJECT with *obj_type*=KC_TACCLASS:
Determine current setting
KC_MODIFY_OBJECT with *obj_type*=KC_TACCLASS:
change number of processes

Changing the setting for timers

Timers are defined to prevent processes from remaining assigned for excessive periods of time while waiting for resources to be freed up or for connections and sessions to be established. The timers monitor these wait times and roll back the waiting transaction after the specified time elapses. The timers are defined during KDCDEF generation and can be adapted at runtime.

In openUTM, timers are defined for the following wait times:

- Wait time after a blocking call (*pgwttime*)
- The timer monitors the maximum length of time which a program unit waits before returning to the program unit after placing a blocking call.
- Maximum length of time during a transaction that is spent waiting for an answer from a dialog partner (*termwait*...).
- Maximum period of time over which resources can remain assigned by a transaction and the maximum period of time that a program unit can wait for resources to be freed up (*reswait*...).

  Using the information functions (parameter type STATISTICS/KC_CURR_PAR) you can, for example, determine how frequently program units have had to wait for locked resources (relative figure).
- Maximum length of time to wait for a session/association to the partner application to be assigned.

The timers are intended as "emergency brakes" for unforeseen situations. You should therefore set the timer values in such a way that they do not run when the application is executing normally. Timeouts should only be caused by exceptional situations, for example when a program error occurs or no response is received from a partner application.

If the timers *pgwttime* or *reswait* are set for an excessively long period, particularly in bottleneck situations, then individual processes in the application can be assigned by program units which either lock resources for too long at a time (long-running units) or wait

too long for required resources to become free. However, if the timers are not set for long enough periods, system performance is impaired by transactions being rolled back frequently.

KDCINF SYSPARM or STATISTICS:
Determine current timer settings and request information about current wait times
KDCAPPL: change timer setting

KC_GET_OBJECT with *obj_type*=KC_TIMER_PAR / KC_CURR_PAR:
Determine current timer settings and request information about current wait times
KC_MODIFY_OBJECT with *obj_type*=KC_TIMER_PAR: change timer setting

Restricting the number of users/clients signed on

At runtime you can influence the number of users/clients that can connect to the application and request services from the application at the same time. For this purpose, you are offered the following options:

- You can restrict the total number of users/clients able to sign on to an application at the same time.
- You can restrict the number of clients able to connect via individual LTERM pools at the same time. To do this, you disable some of the LTERM partners in the pool.
- You can disable individual clients/LTERM partners/users.
- You can disable LTERM pools completely. At this point, it is no longer possible for users/clients to sign on to the application via a disabled LTERM pool.

B

- You allow only a small number of parallel sessions access to a multiplex connection.

KDCAPPL MAX-CONN-USERS: total number of users/clients
KDCPOOL: disable a number of approved pool LTERM partners / LTERM pool

KC_MODIFY_OBJECT
obj_type=KC_MAX_PAR: define total number of users/clients
obj_type=KC_TPOOL:
disable a number of approved pool LTERM partners / LTERM pool
obj_type=KC_PTERM: disable clients/printers
obj_type=KC_LTERM: disable LTERM partners
obj_type=KC_USER: disable users

Disabling services

It is, for example, possible to disable long-running services for a certain period by disabling the relevant transaction code (State OFF). As of this point, jobs are no longer accepted for disabled transaction codes. In the case of disabled asynchronous TACs, no further jobs are written to the message queue either.

You can disable a transaction code either exclusively as a service TAC or as both a service TAC and a follow-up TAC (complete lock: State STOP).

You can also lock asynchronous services using the KEEP status, which means that jobs for the asynchronous TAC are accepted, but not processed immediately. They can subsequently be processed when the application is less busy, e.g. at night.

KDCTAC

KC_MODIFY_OBJECT *obj_type*=KC_TAC

Preventing or remedying bottlenecks for connections to partner applications

If bottlenecks occur during communication with LU6.1 or OSI TP partner applications, you can perform the following actions:

- Establish other transport connections to an LU6.1 partner application.
- Before you can communicate with a partner application, you must first have created or generated several parallel connections, but not all the connections created or generated should yet have been established.
- Increase the number of parallel logical connections to an OSI TP partner application. The maximum possible number of parallel connections is defined during generation in the OSI-LPAP statement.
- Adapt the timer (access wait) for the wait time following a request for a remote service within which a session or association with a partner application is to become available or be established. You can set this timer individually for each LTAC. If the timer is set to 0 for an asynchronous LTAC, asynchronous jobs for this LTAC are also not arranged in the local message queue of the partner application.
- Adapt the timer (reply wait) which monitors the wait time for an answer from the partner application. This timer is also set individually for each LTAC.
- Adapt the setting of the idle timer. This timer indicates the length of time that a session or association can remain unused before openUTM terminates the connection to the partner application. If the timer setting is too long, an inordinate number of resources will be reserved by unnecessary connections. If the timer setting is too short, too may resources will be used up to allow the connection to be set up again. The timer is set individually for each partner.

X/W X/W X/W X/W X/W If a large amount of connections in your application are handled by the same BCAMAPPL name or access point in your application, this can give rise to bottlenecks since processes can come up against system limitations (e.g. the maximum number of file descriptors). During the next KDCDEF generation, you should then generate more BCAMAPPL names and access points.

KDCLPAP / KDCLSES: establish connections, adjust idle time
KDCLTAC: change access wait and reply wait

KC_CREATE_OBJECT *obj_type*=KC_CON/KC_LSES:
create connections and sessions

KC_MODIFY_OBJECT
obj_type=KC_LPAP/KC_OSI_LPAP/KC_LSES:
establish connections, adjust idle time
obj_type=KC_LTAC: change access wait and reply wait

Enabling or disabling data compression

If GSSBs, LSSBs, ULS, TLS, or KB program areas are frequently read or written in a length which is greater than one UTM page, you should check whether enabling data compression will enhance the performance of the UTM application.

You can check whether data compression is worthwhile while it is enabled as follows:

KDCINF STAT, *AVG COMPRESS PAGES SAVED* field

KC_GET_OBJECT with *obj_type*=KC_CURR_PAR, *avg_saved_pgs_by_compr* field

3.3 Avoiding a page pool bottleneck

The content and role of the page pool depends on whether the application is a standalone application (see below) or a UTM cluster application (see page 61).

3.3.1 Page pool of a standalone application

User data generated during the application run is stored in the page pool of a standalone application. In addition to UTM memory areas and service data, this includes:

- the message queues of the asynchronous TACs, LTERM, LPAP and OSI-LPAP partners and the user, TAC and temporary queues (i.e. jobs to local services and communication partners and print jobs to the printers of the application) that are not being processed
- dialog jobs or asynchronous jobs buffered for transaction codes of TAC classes, which are interrupted as a result of TAC class control

The page pool size is defined during KDCDEF generation and cannot be modified at runtime.

While an application is running, it is necessary to ensure that the page pool is assigned completely. To this end, two warning levels are defined for KDCDEF generation (page pool assignment in %). If page pool assignment reaches one of these warning levels, openUTM generates message K041. If the destination MSGTAC is defined for this message, you can respond to this event in an MSGTAC routine. If the second warning level (default setting 95%) is reached, no more asynchronous jobs are written to the message queues and no more user log records (LPUT jobs) are written to the user log file. Asynchronous jobs and LPUT calls then are rejected.

For this reason, when the first warning level is reached, measures must be taken to release memory space in the page pool. While the application is running, you can obtain information about the current assignment of the page pool.

KDCINF STATISTICS
KDCINF PAGEPOOL

KC_GET_OBJECT with *obj_type*=KC_CURR_PAR
KC_GET_OBJECT with *obj_type*=KC_PAGEPOOL

However, if page pool bottlenecks occur frequently, the page pool is simply not large enough. In this case, you should regenerate the application and increase the size of the page pool.

The following section describes how to terminate message queues and dialog jobs in buffer storage in order to clear space, i.e. relieve congestion, in the page pool.

Reducing the size of message queues

You can implement the following measures to reduce the size of message queues:

- Reduce printer queues by establishing connections to all printers for which print jobs are waiting. These print jobs will then be processed immediately even if a control value (*plev*) has been generated for a printer and this has not yet been reached.
- Request connections to TS applications and partner applications for which asynchronous jobs are in buffer store in the page pool. If the communication partners are disabled, they must first be re-enabled.
- Increase the number of processes that can be used concurrently for asynchronous processing purposes.
- Increase the number of processes that can be used concurrently for processing jobs of a specific TAC class (in applications without priority control).
- Unlock (status ON) or lock with status OFF any asynchronous transaction codes and TAC queues that are locked with the KEEP status or blocked. The KEEP status means that jobs for the transaction code or queue in question are accepted, but are not processed immediately, whereas the status OFF means that no further jobs are accepted, but any waiting jobs will be processed.
- Delete the asynchronous jobs in the message queues of dynamically deleted LTERM partners and asynchronous TACs.
- Delete older messages from service-controlled queues if they are no longer expected to be read.
- Assign messages from the dead letter queue to a new destination again in order to allow them to be edited.

KDCINF STATISTICS:
total number of all messages in the buffer store in the page pool

KDCINF LTERM / LPAP / OSI-LPAP / TAC:
query the assignment of message queues for individual objects

KDCINF PAGEPOOL:
query the page pool page utilization subdivided according to types

KDCAPPL SPOOLOUT: reduce size of printer queues

KDCLTERM or KDCLPAP: establish connection to communication partners

KDCAPPL ASYNTASKS: change the number of processes

KDCTAC STATUS: change the status of a transaction code

KDCTCL: change the number of processes in a TAC class

KC_GET_OBJECT with *obj_type*=KC_CURR_PAR:
query the total number of messages in buffer store in the page pool
with *obj_type*=KC_LTERM / KC_LPAP / KC_OSI-LPAP / KC_TAC:
assignment of message queues of individual objects
with *obj_type*=KC_PAGEPOOL:
query the page pool page utilization subdivided according to types

KC_SPOOLOUT: reduce the size of printer queues

KC_MODIFY_OBJECT
with *obj_type*=KC_LTERM/ KC_LPAP/KC_OSI_LPAP: establish connections
with *obj_type*=KC_TASKS_PAR: change number of ASYNTASKS processes
with *obj_type*=KC_TAC: change the status of a transaction code or a TAC queue
with *obj_type*=KC_TACCLASS: change the number of processes in a TAC class

DADM (KDCS call): delete jobs and move messages from the dead letter queue

In applications without TAC-PRIORITIES:
reducing the size of job queues in TAC classes

The information functions enable you to determine the number of jobs in buffer storage in the page pool in any given TAC class. The information which openUTM issues on a TAC class includes the number of messages stored in buffer storage in the page pool.

In order to reduce the size of these queues you can increase the maximum number of processes able to process jobs in this TAC class at the same time.

KDCINF TACCLASS query number of dialog jobs in buffer storage
KDCTCL: change number of processes

KC_GET_OBJECT with *obj_type*=KC_TACCLASS:
query number of dialog jobs in buffer storage

KC_MODIFY_OBJECT with *obj_type*=KC_TACCLASS:
change number of processes

Enabling or disabling data compression

When a large number of page pool pages are utilized for GSSBs, LSSBs, TLS, or ULS (KDCINF PAGEPOOL or KC_GET_OBJECT with *obj_type*=KC_PAGEPOOL), you should check whether enabling data compression might possibly reduce the number of utilized pages.

You can check whether data compression is worthwhile while it is enabled as follows:

KDCINF STAT, *AVG COMPRESS PAGES SAVED* field

KC_GET_OBJECT with *obj_type*=KC_CURR_PAR, *avg_saved_pgs_by_compr* field

3.3.2 Page pools of a UTM cluster application

Every node application in a UTM cluster application has its own page pool for data that is local to the node. In addition, there is a common cluster page pool for data that is valid globally throughout the cluster. This results in certain special characteristics compared to standalone applications:

- Data that applies locally to the node is stored only in the page pool of the relevant node application. Data that applies locally in the node includes, for example, the TLS areas, message queues as well as buffered dialog or asynchronous jobs to transaction codes of TAC classes which have been interrupted due to TAC class control activities.
- Data that applies globally throughout the cluster is stored in the cluster page pool. This type of data includes GSSB, ULS or cluster-wide service data.

Properties of the cluster page pool

The cluster page pool forms part of the UTM cluster files and consists of a management file and one or more files containing the user data. The following are defined during generation with KDCDEF:

– The size of the cluster page pool file(s)
– The number of cluster page pool files
– A warning level for the cluster page pool

The message that the value has risen above or fallen below the warning level is always output by the node application that triggered the change of state.

The administration functions permit the following actions:

- You can determine the current occupancy of the cluster page pool and reset the statistical values, e.g. by means of WinAdmin, WebAdmin or the KDCADMI program interface.

  KC_GET_OBJECT and KC_MODIFY_OBJECT with *obj_type*=KC_CLUSTER_CURR_PAR

- You can increase the size of the cluster page pool files without terminating the UTM cluster application.

  openUTM manual “Using openUTM Applications”, entry for "Increasing the size of the cluster pagepool" in the section "Update generation in a cluster".

3.4 Replacing the application program

You can use the administration functions of openUTM to replace the entire application program or parts of the application program (individual load modules or shared objects) without having to terminate the application.

In order to replace individual parts of the application program, the application program must have been generated with load modules (on BS2000 systems) or with shared objects (on Unix or Linux systems) or DLLs (on Windows systems).

For more detailed information about program replacement and the conditions governing program replacement, see the openUTM manual “Using openUTM Applications”.

KDCAPPL PROGRAM: replacement of the entire application program
KDCPROG: replacement of individual load modules, shared objects or DLLs

KC_CHANGE_APPLICATION: replacement of the entire application program
KC_MODIFY_OBJECTS with *obj_type*= KC_LOAD_MODULE:
replacement of individual load modules, shared objects or DLLs

B B In standalone UTM applications, please proceed as follows when replacing load modules stored in a common memory pool:

B B B B 1. Identify the load modules to be replaced. To do this, call KC_MODIFY_OBJECT with *obj_type*= KC_LOAD_MODULE for these load modules and indicate which version is to be loaded during the ensuing replacement operation. Alternatively, you can use the KDCPROG command.

B B B 2. In order to replace the identified load modules, the entire application program must be terminated (all individual processes) and reloaded. To do this, you call KC_CHANGE_APPLICATION or use the KDCAPPL command.

B B In UTM cluster applications, the replacement of the entire application program is initiated immediately when the version of a load module is changed.

3.5 Clients and printers

For clients and printers in an openUTM application, you can perform the actions described in the following section.

W Printers are not supported by openUTM on Windows systems.

Transferring logical properties from one terminal to another

If a terminal is defective, or if the user previously connected to the terminal wishes in future to work from a different terminal, you can transfer the logical properties of one terminal to another one in stand-alone UTM applications. You do this by assigning the LTERM partner of one terminal to another terminal (of the same type). In so doing, you can for example transfer the following properties to the new terminal:

– restart information
– access rights (key set)
– access protection (access list or lock code)
– message queue with asynchronous messages
– user ID for the automatic KDCSIGN, where defined
– language environment, where defined
– start format, where defined
– control value *qlev* for the message queue, where defined

KDCSWTCH

KC_MODIFY_OBJECT with *obj_type*=KC_PTERM

Assigning the message queue of one printer to another printer

In standalone UTM applications, if one printer malfunctions, the printer queue can be assigned to another printer (of the same type). This printer then processes the print jobs in that queue. To do this, you must disable the defective printer and assign the LTERM partner of the printer to a different one.

In addition to the printer queue, defined logical properties are also transferred to the new printer. This includes the control value *qlev* for the printer queue and the value *plev*. As soon as plev print jobs are waiting in the printer queue, openUTM automatically sets up a connection to the printer.

KDCPTERM: Disable a printer
KDCSWTCH: Assign an LTERM partner to a different printer

KC_MODIFY_OBJECT with *obj_type*=KC_PTERM

Generating printer pools

In standalone UTM applications, at runtime you can group printers in the application together into printer pools. Printer pools are created when you assign additional printers to the LTERM partner of one printer. The printer queue belonging to the LTERM partner is then processed jointly by all printers assigned to that LTERM partner. Good reasons for generating a printer pool can include:

- The message queue of a printer may become too large. It may prove necessary to wait too long for requested print outputs and the page pool in which jobs are kept in buffer storage can be placed under excessive strain. To process print jobs in the queue, several printers should be implemented.
- When a printer is entered, if the maximum specified number of print jobs which can be stored in a printer queue at one time (*plev*) is too small, print jobs sent to this printer will be rejected frequently.
- Additional printers have recently become available in a branch office. These printers are to process all print jobs from this branch office on a joint basis, i.e. when a print job is issued, it is sent for processing to a printer which is free at the time. You can load these new printers in the configuration dynamically and group them in printer pools with the existing printers.

KDCSWTCH

KC_MODIFY_OBJECT with *obj_type*=KC_PTERM

Disabling printers/clients and their LTERM partners

You can disable clients and printers and their LTERM printers. It is not possible to establish a connection to disabled clients or via disabled LTERM partners. You can still send asynchronous jobs to disabled LTERM partners. These are then stored in the message queue until the control value for that message queue is reached. However, the jobs are not processed until the LTERM partners are re-enabled.

KDCLTERM, KDCPTERM

KC_MODIFY_OBJECT with *obj_type*=KC_PTERM or KC_LTERM

Connections to clients and printers

If necessary you can establish and terminate connections to TS applications, terminals and printers.
In the case of terminals, TS applications and printers that are always connected to the application, you can arrange for connections to be established automatically each time the application starts.

KDCLTERM, KDCPTERM

KC_MODIFY_OBJECT with *obj_type*=KC_PTERM or KC_LTERM

Reading information about the availability of clients and printers

Using the information functions of openUTM you can query information about the availability of clients and printers. The following information is provided:

- Current status of client/printer (is it disabled at present or not?)
- Does a connection exist at present, or is an attempt currently being made to establish a connection?
- Period of time where the printer or client has already been connected to the application
- Number of messages replaced on the connection
- Number of failures in the connection to client/printer
- Control value of message queue (*qlev*)
- Number of jobs in the message queue of a printer/printer pool for which a connection to the printer (pool) is established automatically.

KDCINF LTERM or PTERM

KC_GET_OBJECT with *obj_type*=KC_PTERM or KC_LTERM

4 Changing the configuration dynamically

openUTM provides you with functions at the administration program interface with which you can create new objects in the configuration or delete them from the configuration during application runtime.

These functions further increase the availability of UTM applications. Regeneration of the application with KDCDEF, for which operation has to be interrupted, is now required much less frequently. In addition, regeneration of a UTM application is now much easier and a great deal less time-consuming. You will find appropriate recommendations for regenerating a UTM application in section “Recommendations for regeneration of an application” on page 107.

Using the functions UTM provides for changing the configuration dynamically, you can create and delete the following objects:

– user IDs, including the associated queues
– key sets
– transport connections to remote LU6.1 applications
– LU6.1 sessions
– transaction codes for your own application
– transaction codes, via which service programs can be started in partner applications
– LTERM partners
– clients, printers
– program units and VORGANG exits
  (only in applications with load modules, shared objects or DLLs)
– TAC queues

To add and delete objects, use either the administration tools WinAdmin and WebAdmin or administration programs you have generated yourself. Using the KC_CREATE_OBJECT call at the administration program interface, you can add new objects to the configuration. With the KC_DELETE_OBJECT call, you can delete objects from the configuration. The KC_MODIFY_OBJECT call allows you to change individual object properties.

The full range of functions for dynamically changing the configuration can also be used in the function variant UTM-F. openUTM saves all the changes made to the configuration (including the entry, deletion and modification of dynamic objects) in the KDCFILE. The modified configuration data is then available for the next application run.

The following section describes a number of things you need to be aware of during KDCDEF generation of the application if you wish to add or delete objects to/from the configuration at runtime. It also describes points you must consider when dynamically creating objects from your application configuration.

4.1 Requirements for KDCDEF generation

To enable you to add objects dynamically to the configuration of your UTM application, you must make the following preparations when generating the application with KDCDEF.

No preparations are required for deleting objects from the configuration during KDCDEF generation.

Reserving spaces in the object tables of the KDCFILE

The configuration data of a UTM application is stored in the object tables of the KDCFILE that is created during KDCDEF generation of the application. During KDCDEF generation, the space required to accommodate these tables is also defined. For this reason, during KDCDEF generation, you must reserve table spaces for any objects which you wish to add to the configuration of your application at runtime. You are assisted in this process by the KDCDEF statement RESERVE (see the openUTM manual “Generating Applications”).

In the RESERVE statement you indicate how many table spaces are to be set aside for each single type of object, i.e. how many LTERM partners are to be created dynamically, how many transaction codes etc. Table spaces are reserved individually for each object type, i.e. a table space which you have reserved for an LTERM partner cannot be occupied by a transaction code etc.

During the application run, you can dynamically create as many objects of one type as you have reserved table spaces with KDCDEF. Deleting another object of the same type does not free up a table space for a new object. An exception to this are user IDs and connections for distributed processing by means of LU6.1 for stand-alone applications. These you can delete from the configuration immediately (see section “Deleting objects dynamically from the configuration” on page 85). The table spaces occupied by these user IDs or LU6.1 connections are then freed up immediately and are thus available for new user IDs and LU6.1 connections.

When reserving table spaces with RESERVE, always consider the following points:

openUTM internally creates one user ID for each UPIC and for each TS application (client of type APPLI or SOCKET) which you add dynamically to the configuration. In UTM applications generated with user IDs (i.e. where KDCDEF generation contains at least one USER statement), an additional table space is reserved for user IDs for every APPLI, SOCKET or UPIC-R type client created dynamically. These table spaces are not freed up, when clients are deleted. In applications with no user IDs, these table spaces are reserved by openUTM internally.

For further information about reserving table spaces, see the openUTM manual “Generating Applications”, RESERVE control statement.

Generating lock codes, BCAMAPPL names and the formatting system

In the KDCDEF run you must have already generated objects or values statically in advance if you want to reference them later in dynamic configuration; examples of this are the value range of lock codes and the names of the transport system access points of the local application.

- Lock codes (access protection) which you wish to assign to the transaction codes and LTERM partners must fall in the range between 1 and the maximum value defined in KEYVALUE (MAX statement). For this reason, you should select a sufficiently high number for KEYVALUE and also generate keysets containing the appropriate keycodes (see notes on the lock/keycode concept in the openUTM manual “Concepts und Functions”).
- All names in the local application (BCAMAPPL names) which are to be set up using connections to clients or printers must be generated using KDCDEF. In particular, remember that you have to generate special BCAMAPPL names in order to link TS applications through the socket interface (PTYPE=SOCKET).

B B B
- If start formats are to be assigned to user IDs and LTERM partners, a formatting system must be generated during KDCDEF generation (FORMSYS statement). If #formats are used as start formats, an additional sign-on service must be generated.

Requirements for adding program units and VORGANG exits

You can only add new program units and VORGANG exits to the configuration of your application dynamically if the application satisfies the following requirements:

B B
- UTM applications on BS2000 systems must be generated with load modules (KDCDEF generation with LOAD MODULE statements).

X X
- UTM applications on Unix or Linux systems must be generated with shared objects (KDCDEF generation with SHARED-OBJECTS statements).

W W W
- UTM applications on Windows systems must use Windows DLLs. You will find further details on how to generate the application in the openUTM manual “Generating Applications”.

A program unit which you wish to create dynamically at runtime must be linked to a load module, shared object or a DLL which was defined during KDCDEF generation.

B B
However, the program unit should not be linked to a load module which is linked statically to the application program (STATIC load mode)

At least one program unit must have been generated with KDCDEF for each programming language in which you wish to create program units in your application. Only then does the application program contain the language link modules and runtime systems it requires in order to run.

B B B In the case of program units compiled with ILCS-capable compilers (COMP=ILCS), it is sufficient to generate a program unit with COMP=ILCS during KDCDEF generation. No PROGRAM statements have to be submitted for the various programming languages.

B B B In the case of COBOL programs, the relevant LOAD-MODULE must be generated with ALTERNATE-LIBRARIES=YES in order to allow the required RTS modules to be dynamically loaded by autolink.

Requirements for the dynamic creation of transaction codes

If you wish to add transaction codes dynamically to the configuration, you must take account of the following points:

- Transaction codes for program units which use an X/Open program interface can only be created dynamically if at least one transaction code for an X/Open program unit was generated statically with KDCDEF (TAC statement with API≠KDCS).
- If you wish to divide the transaction codes into TAC classes, in order to be able to control job processing, then you must create at least one TAC class during KDCDEF generation.

  During KDCDEF generation you can create TAC classes in three ways:

  1. Generate a transaction code for which you specify a TAC class in the TACCLASS operand (TAC statement). KDCDEF will then implicitly generate the specified TAC class.
  2. If you are running the application without priority control (it contains no TAC-PRIORITIES statement), you can generate TAC classes by writing a TACCLASS statement.
  3. You can create TAC classes implicitly by writing a TAC-PRORITIES statement.

  Once you have created a TAC class during KDCDEF generation you can assign the transaction codes which you create dynamically to any TAC class of your choice between 1 and 8 (dialog) or 9 and 16 (asynchronous). The TAC classes are created by openUTM implicitly. These implicitly created TAC classes can be administered.

  If you generated the application without TAC-PRIORITIES, openUTM specifies the number of processes (TASKS) in implicitly generated TAC classes as follows:
  1 for dialog TAC classes (classes 1 to 8),
  and 0 for asynchronous TAC classes (classes 9 to 16).

  However, openUTM only creates asynchronous TAC classes if you set ASYNTASKS > 0 in the MAX statement during KDCDEF generation.

In applications containing TAC classes without priority control, you can only create transaction codes dynamically which start program unit procedures with blocking calls if TAC classes with PGWT=YES (dialog and/or asynchronous TAC class) were explicitly created with TACCLASS statements in KDCDEF generation and MAX TASKS-IN-PGWT > 0.

- In applications with priority control (with TAC-PRIORITIES statement), you can only create transaction codes dynamically which start program unit procedures with blocking calls (*kc_tac_str.pgwt*=´Y´) if MAX TASKS-IN-PGWT>0 was specified during KDCDEF generation.

Requirements for the dynamic creation of user IDs

You can only add user IDs to the configuration dynamically if your application was generated with user IDs. For this, your KDCDEF generation must contain at least one USER statement and at least one user ID must have administration privileges (USER with PERMIT=ADMIN).

B/X
B/X
B/X
B/X

If new user IDs with ID cards are also to be added to the configuration at runtime then, when reserving table spaces with the RESERVE statement, you must explicitly indicate what percentage of user ID table spaces is to be set aside for user IDs with ID cards (CARDS operand in the RESERVE statement).

B
B

If user IDs with Kerberos authentication are to be dynamically generated during operation, they must be reserved using the PRINCIPALS operand of the RESERVE statement.

4.2 Adding objects to the configuration dynamically

Using the KC_CREATE_OBJECT call you can add new objects to the configuration of your application during an application run.

KC_CREATE_OBJECT on page 188

You can create exactly one object per KC_CREATE_OBJECT call. However, within the administration program, you can call KC_CREATE_OBJECT several times in order to create several objects. When you place a call, you indicate the type of object, its name and the properties you wish the object to have.

The creation of objects is subject to transaction management. Configuration data is not written to the object table until the transaction has been logged successfully. This means that an object created in a program unit cannot be accessed until the transaction has been concluded successfully. The object cannot be used before this happens and it is also not possible to read or modify the object's properties. Calls such as KC_MODIFY_OBJECT or KC_GET_OBJECT can be submitted for the new object only after successful completion of the new create operation, i.e. after successful completion of the transaction.

During the transaction in which an object is created, access to this object is only permitted in order to establish a relational link to another object created in the same transaction. For example, a relationship of this kind can be established between a client or printer and its connection point, the LTERM partner, between a transaction code and the related program unit, between a transaction code and its VORGANG exit, or between a key set and the objects (such as LTERMs, USERs, TACs or LTACs) to which it refers.

If two objects which relate to one another are created in one transaction, you must pay careful attention to the order in which the objects were created. For example, you can create a client together with its connection point (LTERM partner) in one and the same transaction. However, the LTERM partner must be created before the client since the name of the LTERM partner is indicated when the client is created.

As a general rule, all objects to which you refer when creating a new object must either already feature in the configuration or have been created in the same transaction prior to the new object. The following section provides a detailed description of each type of object showing the sequence in which the objects must be created.

UTM cluster applications

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. the objects are dynamically entered in the configuration in all the node applications.

Availability of dynamically created objects

Dynamically created objects are a component of the configuration, even in subsequent application runs, unless they were deleted with KC_DELETE_OBJECT. The same applies to objects in a UTM-S and a UTM-F application.

4.2.1 Adding clients, printers and LTERM partners

To add a client or printer you must call KC_CREATE_OBJECT with object type KC_PTERM. To add an LTERM partner, you must specify object type KC_LTERM.

W
W

Printers are not supported in UTM applications running on Windows systems.

To enable you to connect a client or printer to the application, an LTERM partner must be assigned to it. If you specify this LTERM partner when adding a client or printer, the LTERM partner must either already exist in the configuration of that application or have been created in the same transaction prior to the client/printer. The following rule therefore applies:

LTERM partner (KC_LTERM) before client/printer (KC_PTERM)

When adding clients/printers, you must distinguish between the following two cases:

– terminals and printers
– TS applications and UPIC clients

Terminals and printers

You can add terminals and printers to the configuration without assigning an LTERM partner directly to them, i.e. you do not have to specify an LTERM partner when adding them. You can then assign the LTERM partner to the terminal or printer at a later date. To do this, you are provided with the administration command KDCSWTCH and the call KC_MODIFY_OBJECT (object type KC_PTERM). Actual assignment must then take place in a separate transaction.

However, if you do specify an LTERM partner when adding a terminal or printer then, according to the rule stated above, this LTERM partner must already exist in the configuration of that application or have been created in the same transaction as the terminal or printer before the terminal or printer was added.

You can assign an LTERM partner to a printer even if the LTERM partner is already assigned to another printer. This does not cancel the previous assignment. One LTERM partner can be assigned to a number of printers. These printers then form a printer pool and process the message queue of the LTERM partner jointly.

You can only assign an LTERM partner which is not already assigned to another client. Any assignment to another terminal which already exists must be cancelled before the client is created in a separate transaction (with the administration command KDCSWTCH or the call KC_MODIFY_OBJECT).

If an LTERM partner is to be created explicitly with an automatic KDCSIGN to connect a terminal, you must, during the create operation, assign the user ID under which the automatic KDCSIGN is to be executed when a connection is being established. The user ID must already feature in the configuration before the LTERM partner is added, or have been created in the same transaction before the LTERM partner. Generally speaking, the following rule applies:

User ID (KC_USER) before LTERM partner (KC_LTERM)
before terminal (KC_PTERM).

As a general rule, the following applies:
The property *usage_type* (D for dialog partner or O for output medium) of the LTERM partner must match the value which you specify in *usage* when adding the client/printer.

If an LTERM partner is created for a printer which is to be administered by a print control LTERM (CTERM), you must assign the printer control LTERM when adding the LTERM partner. Before adding the LTERM partner, the printer control LTERM must either already be in the configuration of the application (created statically or dynamically) or in the same transaction as the LTERM partner, where it must have been created before the LTERM partner. The following rule applies:

Printer control LTERM (KC_LTERM) before LTERM partner (KC_LTERM)
before printer (KC_PTERM)

TS applications and UPIC Clients

You must assign an LTERM partner when creating TS applications or UPIC clients (APPLI, SOCKET, UPIC-R or UPIC-L type clients). This LTERM partner must be added in the same transaction as the client but before the client itself. In other words, the KC_CREATE_OBJECT call which creates the LTERM partner must be processed in the same transaction and before the KC_CREATE_OBJECT call which creates the client. In this instance, the rule to apply is as follows:

LTERM partner (KC_LTERM) before the TS application/UPIC client (KC_PTERM) in the same transaction

The assignment of a client to an LTERM partner cannot be cancelled as long as the client remains in the configuration.

For the LTERM partner of a client of this type, openUTM requires a permanently assigned user ID, i.e. the connection user ID.

You can create a connection user ID explicitly, in which case it has to be included in the same transaction as the LTERM partner and the client. However, the user ID must be added to the configuration *before* the client. When assigning a user ID to an LTERM partner, you must distinguish between the following cases:

- You are explicitly creating a user ID with the name of the LTERM partner. In this case, assignment is automatic when you add the LTERM partner.
- You are creating a user ID with any name. In this case, you must explicitly enter the name when adding the LTERM partner (field *kc_lterm_str.user_gen*).

If you do not create the connection user ID explicitly, openUTM implicitly creates a user ID with the name of the LTERM partner.

The connection user ID is always reserved for this client. No other user or client can log on with the application under this user ID.

The user ID is assigned one of the reserved table spaces. If there are no more spare table spaces for this user ID, the LTERM partner and client are not added to the configuration. The KC_CREATE_OBJECT calls are then rejected.

In general terms, the following applies:
In applications with user IDs, you need three reserved table spaces to add a client of type APPLI, SOCKET or UPIC-R/UPIC-L: one for object type PTERM, one for object type LTERM and one for object type USER.

The following sequence must be observed:

User ID (KC_USER) before LTERM partner (KC_LTERM) before
TS application/UPIC client (KC_PTERM)
All three objects must be created in the same transaction

A connection user ID cannot be administered, i.e. once you have created the user ID, you can no longer modify its properties.

Example of creating a TS application or an UPIC client

A program which creates a TS application or an UPIC client and which explicitly assigns it a connection user ID must have the structure illustrated in the diagram below. The KDCS calls in angle brackets are optional. The individual KC_CREATE_OBJECT calls, in particular, can be located in various different KDCS programs. However, these programs must run in the same transaction (terminate program, for example with PEND PA).

```
#include <kcadminc.h>                    /* Record definitions              */
INIT                                     /* KDCS call for signing on with   */
                                         /* UTM
[MGET]                                   /* KDCS call for reading the       */
                                         /* calling TACs and the            */
                                         /* passing parameters              */
KC_CREATE_OBJECT with obj_type=KC_USER   /* KDCADMI call for creating the   */
                                         /* user ID                         */
/* Possible error handling: the following KC_CREATE_OBJECT call should    */
/* only be submitted if the previous call was error-free.                 */

KC_CREATE_OBJECT with                    /* KDCADMI call for creating the   */
obj_type=KC_LTERM                        /* LTERM partner                   */
/* Possible error handling                                                */

KC_CREATE_OBJECT with                    /* KDCADMI call for creating the   */
obj_type=KC_PTERM                        /* client                          */
/* Possible error handling                                                */
MPUT                                     /* KDCS call for sending a message */
....                                     /* to the job-submitting service   */
PEND FI / RE / SP / FC                   /* KDCS call to terminate the      */
                                         /* transaction                     */
```

4.2.2 Adding program units, transaction codes, TAQ queues and VORGANG exits

To add a new program unit or VORGANG exit you must call KC_CREATE_OBJECT for the object type KC_PROGRAM.
When adding a new transaction code or a new TAQ queue, you must specify the object type KC_TAC.

You can only add new program units and VORGANG exits dynamically if the application was generated with load modules (BS2000 systems), shared objects (Unix or Linux systems) or DLLs (Windows systems).

You should assign at least one transaction code to one program unit to enable it to be called. You cannot add the transaction code to the configuration until the program unit has been created. This means that program units must either already be in the application configuration at the time the transaction code is created with KC_CREATE_OBJECT, or they must have been created in the same transaction but before the transaction code was created. The program unit can be created with KDCDEF or may have been created in a separate transaction.

You can also assign new transaction codes to program units already in the configuration.

A newly created program unit cannot be called until it has been loaded and at least one transaction code has been assigned to it. To add the program unit, it must be compiled and linked into the application by a load module, shared object or DLL created with KDCDEF. Following this, this load module, shared object or DLL must be replaced (see KDCPROG on page 764 or KC_MODIFY_OBJECT with *obj_type*=KC_LOAD_MODULE on page 323).

B If the public slice of the load module is located in a common memory pool, you must then still submit a KDCAPPL PROG=NEW or KC_CHANGE_APPLICATION call to arrange for this load module to be replaced. You cannot use the new or modified service until this has been done.

B A new program unit cannot be linked into a load module which is statically linked to the application program (STATIC load mode).

If a VORGANG exit is to be assigned to a transaction code which you are creating dynamically (*kc_tac_str.exit_name*) then this VORGANG exit must exist in the configuration of your application before the transaction code is created or must have been created first (before the code) in the same transaction in which the transaction code itself was created.

To ensure that the VORGANG exit is able to run properly, the relevant program must be created. Dynamically created VORGANG exits must, like program units, be linked to a load module, shared object or DLL which then has to be replaced.

When creating program units, transaction codes and VORGANG exits, the following general rule applies:

Program unit (KC_PROGRAM) and VORGANG exit (KC_PROGRAM)
before transaction codes (KC_TAC)

The transaction codes for the event services BADTAC, MSGTAC and SIGNON (KDCBADTC, KDCMSGTC, KDCSGNTC) cannot be created in the configuration dynamically.

4.2.3 Creating user IDs

When creating a new user ID and an associated USER queue, you must call KC_CREATE_OBJECT for object type KC_USER. User IDs which are to have a fixed assignment to specific LTERM partners for an automatic KDCSIGN must be created before the LTERM partner is added. See also section “Adding clients, printers and LTERM partners” on page 74 for details of things you will need to remember.

4.2.4 Creating key sets

To create a new key set, you have to call KC_CREATE_OBJECT for the object type KC_KSET. You can then assign the new key set in the same transaction to a new user ID, a new LTERM partner, a new transaction code or TAC queue or a new LTAC.

The following rule applies:

Key set (KC_KSET) before LTERM partner (KC_LTERM)
and user ID (KC_USER) and transaction code (KC_TAC) and LTAC (KC_LTAC)

4.2.5 Entering LU6.1 connections for distributed processing

In the case of a link by means of the LU6.1 protocol, for communication between the local UTM application and a remote application you must define one or more transport connections and sessions by means of which the communication relationships are set up.

For the entry of a transport connection, call KC_CREATE_OBJECT for the object type KC_CON. To define a session, call KC_CREATE_OBJECT for the object type KC_LSES.

The prerequisite is that LPAP partners must be known and session properties defined in each application.

A number of CON and LSES objects must be created for each LPAP; the number of CON and LSES objects determines the number of parallel connections that are possible with a partner application via an LPAP.

In cluster applications, it is necessary to generate, for each CON object, as many LSES objects as there are node applications in order to enable the partner application to communicate with all the node applications.

A CON object and an LSES object are created for each parallel connection via an LPAP and assigned to the LPAP. Every CON object and every LSES object in each of the applications involved must be created appropriately so that the following applies:

- A CON name in the local application is the same as a BCAMAPPL name in the remote application and vice versa.
- An LSES name in the local application is the same as an RSES name in the remote application and vice versa.

CAUTION!
It is not permissible for an LPAP name to create a number of CON objects that lead to different applications or are assigned to different LPAPs in the partner application via their corresponding CON objects.

Such configurations are not recognized by UTM and lead to errors when connections and sessions are set up and when sessions are restarted.

4.2.6 Entering LTACs

In order to dynamically create a transaction code for starting a service or a remote service program in a partner application, you have to call KC_CREATE_OBJECT for the object type KC_LTAC.

The local transaction code is assigned either

- the name of a transaction code in a specific partner application (with single-step addressing), in which case the local transaction code addresses both the partner application and the transaction code in this application, or
- the name of a transaction code in any partner application (with double-step addressing). The partner application in which the service program addressed by the local transaction code is to run must be specified explicitly in the program interface.

If access rights are to be granted by means of an access list, the key set used for this must either already exist or be dynamically created beforehand; the dynamic creation of the key set and the referenced LTAC can also take place within a transaction. If the access rights are to be controlled by means of a lock code, the numeric value for the lock code must not be less than 1 or greater than the maximum value permitted in the application (KDCDEF statement MAX, KEYVALUE operand).

The following rule applies:

Key set (KC_KSET) before LTAC (KC_LTAC)

4.2.7 Format and uniqueness of object names

You must assign a name or logical address (clients and printers) to every object which you create dynamically in the configuration using KC_CREATE_OBJECT. Using this name and its logical address, it must be possible to uniquely identify the object in its application. Note the following rules when assigning names.

– You cannot use any reserved names. (→ Reserved names)
– The name of an object must be unique in the class of name belonging to the object name. (→ Unique names and addresses)
– The names must not exceed the specified maximum length and can only contain certain characters (format). (→ Format of the names)

The names of objects tagged for deleting at a later point in time with KC_DELETE_OBJECT may not be used for objects in the same class of name. The names of user IDs and the names of connections for distributed processing by means of LU6.1 that are deleted immediately can be reassigned again immediately.

Reserved names

Names of transaction codes starting with KDC are reserved for transaction codes in the event services and the administration commands. Names starting with KDC **must not** therefore be used for other objects.

B B In UTM applications on BS2000 systems, program unit names must not begin with a prefix that is used for compiler runtime modules (e.g. IT, IC).

X/W X/W In UTM applications on Unix, Linux or Windows systems, names of objects must also not start with KC, x, ITS or mF.

X/W X/W External names (e.g. program unit names) should not begin with ‘f_’, ‘n_’, ‘t_’, ‘a_’, ‘o_’, ‘p_’ or ‘s_’. ‘t_’ is reserved for PCMX. ‘a_’, ‘o_’, ‘p_’ and ‘s_’ are reserved for OSS.

Any names reserved on a specific platform should not be used on any of the other platforms, in order to render the applications portable.

Unique names and addresses

The names and addresses of objects in a UTM application are summarized in name classes. Within each name class, the object names must be uniquely identified. They cannot be assigned to several objects. There are three classes of name:

The following objects belong to the 1st class of names:

- LTERM partners (object type KC_LTERM);
  the LTERM partners of the LTERM pools also belong to this class.
- Transaction codes and TAC queues (object type KC_TAC).
- LPAP and OSI-LPAP partners for the server-server communication (object type KC_LPAP and KC_OSI_LPAP).

The following objects belong to the 2nd class of names:

- User IDs, including the associated queues (object type KC_USER)
- Sessions for distributed processing using LU6.1 (object type KC_LSES)
- Connections and associations for distributed processing using OSI TP (object type KC_OSI_ASSOCIATION)

The following objects belong to the 3rd class of names:

- Clients and printers (object type KC_PTERM).
  In this context, clients are: terminals, UPIC clients, TS applications (DCAM, CMX applications and UTM applications) which do not use LU6.1 and OSI TP protocols for communication.
- Name of the partner application for distributed processing using protocol LU6.1 (object type KC_CON).
- Name of the partner application in the case of distributed processing using the OSI TP protocol.
  Even if it is not possible to generate OSI-CONs dynamically, the names already generated for OSI-CONs are already allocated to this name class and cannot be used for other objects of this name class.

B – Multiplex connections (object type KC_MUX).

The objects listed in the 3rd class of name are communication partners for the UTM application. They or the connections to them must be uniquely identifiable for openUTM. For this reason, every communication partner must be identified with a logical address. The logical address is a name triplet made up of the following components:

1. Name of the communication partner (*pt_name*, *co_name* of the LU6.1 connection, *mx_name*). This is the symbolic name by which the communication partner is known to the transport system.
2. Name of the computer on which the communication partner is located (*pronam*).
3. Name of the local application via which the connection to the communication partner is established (*bcamappl* or ACCESS-POINT). Even if OSI TP connections cannot be generated dynamically, the names that have already been generated for ACCESS POINTS must be taken into account.

Each communication partner must have a different name triplet.

Format of the names

All names which you define must conform to the following conventions:

– The names of LTERM partners, clients and printers (KC_PTERM), transaction codes, user IDs, LU6.1 connections and sessions as well as transaction codes for remote services must only be 1 to 8 characters in length.

– The names of program units can be up to 32 characters in length if the application was generated using load modules/shared objects/DLLs.

B B – Permissible characters for object names in a UTM application on BS2000 systems are: A,B,C,...,Z, 0,1,...,9, #, @, $. Any combination of these characters is permitted.

X/W X/W – Permissible characters for object names in a UTM application on Unix, Linux systems and Windows systems are: A,B,C,...,Z, a, b, c,..., z, 0,1,...,9, #, @, $.

X/W **Mapped host name (Unix, Linux systems and Windows systems)**

X/W X/W X/W X/W Specify unique address information at the KDCADMI interface in order to allow openUTM to communicate with remote partner applications. The address information contains the UTM host name of the partner computer and the port number. You can also make use of host name mapping.

X/W X/W You will find a detailed description of this functionality and of how to proceed in the openUTM manual “Generating Applications” under "Using mapped host names".

4.3 Deleting objects dynamically from the configuration

You can use the KC_DELETE_OBJECT call at the program interface for administration to delete objects from the configuration of your application while the application is running.

KC_DELETE_OBJECT on page 256

We distinguish two methods for deleting objects: delayed delete and immediate delete.

- *delayed delete* (KC_DELETE_OBJECT *subopcode1*=KC_DELAY)

  The term delayed delete is used to mean that objects are simply designated as deleted. The objects and their properties remain in the object table as before. Delayed deletion acts like a permanent lock which cannot be undone. Physical deletion of objects from the object table only takes place during regeneration if you are working with the inverse KDCDEF.

  Users no longer have access to an object designated for delayed deleting. Only the administrator still has read-only access to such objects, i.e. you can read the names and properties of objects designated for “delayed delete” with KC_GET_OBJECT or with the administration command KDCINF. However, it is no longer possible to change the properties of these objects. User IDs designated for a “delayed delete” can, however, be completely removed from the configuration using an “immediate delete”.

  A delayed delete frees up no space in the object table. The names of deleted objects remain assigned, i.e. no more new objects can be created dynamically in their name class. In particular, no new objects can be created dynamically with the same name and the same object type.

  Key sets, LU6.1 sessions, LTACs, LTERM partners, program units, transaction codes and TAC queues can only be removed from the configuration using the delayed delete method.

- *immediate delete* (KC_DELETE_OBJECT *subopcode1*=KC_IMMEDIATE)

  Immediate deletion is only permitted for the user IDs and LU6.1 connections of stand-alone UTM applications.

  Immediate delete removes an object and its properties from the object table with immediate effect. The table space assigned to a user ID or CON object removed using the “immediate delete” method is available for a newly created user ID or CON object right away without the application needing to be regenerated. The name of a user ID or CON object that is deleted immediately does **not** remain locked. You can generate a new user ID or CON object using the same name right away.

  Once an object is deleted in this fashion, nobody, including the administrator, any longer has any kind of access to it, neither read nor write access.

You can delete just one object with each KC_DELETE_OBJECT call (delayed or immediate delete). In any one program unit, you can make several KC_DELETE_OBJECT calls in succession, i.e. you can delete several objects of different types. In the case of objects related to one another, it is nevertheless important to pay attention to the sequence in which these objects are deleted. An object to which other objects are related cannot be deleted until the other related objects have been deleted, i.e. until their relationship has been cancelled by means of administration functions (e.g. KDCSWTCH can be used to terminate the relationship between terminal/printer and LTERM partner). The following sections describe the rules you must observe when deleting objects.

Object deletion, be it delayed or immediate, is subject to transaction management. The object is not deleted until the transaction in which the KC_DELETE-OBJECT is being processed has been completed successfully.

However, only objects that are featured in the configuration can be deleted. In other words, you cannot delete an object created dynamically in the configuration until the transaction in which the create operation took place has been completed.

Deletion in UTM-F and UTM-S applications applies beyond the end of these applications and cannot be undone.

UTM cluster applications

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. objects are deleted from the configuration in all the node applications. Only delayed deletion is permitted in UTM cluster applications.

4.3.1 Deleting clients/printers and LTERM partners

Clients/printers and LTERM partners can only be removed from the configuration with a delayed delete.

To delete a client or printer from the configuration you must call KC_DELETE_OBJECT (with *subopcode1*=KC_DELAY) for the object type KC_PTERM. To delete an LTERM partner, you have to indicate the object type KC_LTERM.

You are only allowed to delete a client/printer and its related LTERM partners if the client/printer is not connected to the application. For this reason, you should disable the client/printer before deletion to prevent errors from occurring. Such disabling operations must take place in a separate transaction. To disable the client/printer, see KDCPTERM on page 769 or KC_MODIFY_OBJECT with *obj_type*=KC_PTERM as of page 343.

Clients/printers and their associated LTERM partners have a logical relationship to one another. For this reason, you must pay attention to the sequence when deleting clients, printers and their LTERM partners. In general terms, the following rule applies:

An LTERM partner cannot be deleted while a.client/printer is assigned to it.

If the client/printer and the related LTERM partner are to be deleted from the configuration, the following rule applies:

Client/printer (KC_PTERM) before LTERM partner (KC_LTERM).

Both objects can only be deleted from the configuration one after the other in different transactions.

When deleting LTERM partners, please note:

- With UPIC clients (type UPIC-R and UPIC-L) and TS applications (type APPLI or SOCKET), you must delete the client from the configuration before deleting the LTERM partner.
- With terminals and printers, you can delete the LTERM partner without removing the terminal or printer from the configuration. In this event, before deleting the LTERM partner, you must assign the client or printer to another LTERM partner in a separate transaction (KDCSWTCH on page 785 or KC_MODIFY_OBJECT with *obj_type*=KC_PTERM on page 343).

You cannot delete the following LTERM and PTERM partners:

– LTERM partners belonging to an LTERM pool
– LTERMs, belonging to LTERM bundles or LTERM groups,
– printer control LTERMs
– the LTERM partner KDCMSGLT which openUTM creates internally for the MSGTAC service
B – LTERM partners belonging to a multiplex connection
– LTERM and PTERM partners that are used for cluster-internal communication in UTM cluster applications.

You can delete all other LTERM partners and clients/printers from the configuration if you comply with the above rules, regardless of whether they were added to the configuration statically (with KDCDEF) or dynamically.

You can delete the LTERM partner defined as recipient (*destadm*) for the results of asynchronous administration commands. However, in this case, you should define a new recipient, as otherwise the results of asynchronously processed administration commands are lost. To do this, you have the KC_MODIFY_OBJECT call with parameter type KC_MAX_PAR and the administration command KDCAPPL.

Deleting clients, printers and LTERM partners has the following effects:

- It is no longer possible to set up a connection to a deleted client/printer. This means that no more messages can be sent to a client or printer once it has been deleted.
- No more asynchronous messages can be created for a deleted LTERM partner. In other words, no more asynchronous jobs can be added to the message queue of the LTERM partner.
- Asynchronous jobs in the message queue of the LTERM partner at the time of deletion, i.e. jobs created before the deletion process, can no longer be read from the queue by the client/printer. In other words, the asynchronous jobs in the queue can no longer be processed. However, they can still be accessed by administration functions: they can be deleted from the queue. To do this, you can use the KDCS call DADM (see openUTM manual „Programming Applications with KDCS”).
- Asynchronous jobs created by an LTERM partner which has already been deleted are still able to run and can be administered. However, when processing jobs, it is no longer possible to create any further asynchronous jobs (follow-up jobs).
- TLS areas (TLS = terminal-specific long-term storage area) belonging to a deleted LTERM partner are still available for read and write accesses.

4.3.2 Deleting program units, transaction codes and VORGANG exits

Program units, transaction codes, TAC queues and VORGANG exits can only be deleted from the configuration using the delayed delete method.

To delete a program unit or VORGANG exit from the configuration you must call KC_DELETE_OBJECT (with *subopcode1*=KC_DELAY) for the object type KC_PROGRAM. To delete a transaction code or a TAC queue, you must specify the object type KC_TAC.

Transaction codes and the program unit to which this transaction code is assigned are related to one another. In the same way, a VORGANG exit is related to the transaction codes to which it is assigned. For this reason, you must note the sequence followed when deleting transaction codes, program units and VORGANG exits. The following rule applies:

A program unit/VORGANG exit cannot be deleted until all related transaction codes have been deleted.

The following program units must not be deleted:

- program units belonging to the event exits, START, SHUT, FORMAT or INPUT.
- B program units and VORGANG exits linked to load modules with the STATIC load mode.
- X/W X/W X/W program units and VORGANG exits linked statically to the application program, i.e. you can only delete program units and VORGANG exits that are contained in shared objects or DLLs.

The following transaction codes must not be deleted:

- transaction codes KDCMSGTC, KDCSGNTC, KDCBADTC in event services MSGTAC, SIGNON and BADTACS
- the administration command KDCSHUT in the administration program KDCADM
- transaction codes KDCTXCOM and KDCTXRLB created internally by openUTM for XATMI.
- Transaction codes defined in the SIGNON-TAC parameter of the BCAMAPPL statement.

The following TAC queue must not be deleted:

- the dead letter queue KDCDLETQ.

You can delete all other program units and VORGANG exits (that are not statically linked) and transaction codes from the configuration, regardless of whether they were created in the configuration dynamically or statically.

You can delete an asynchronous TAC or a TAC queue defined as a recipient (*destadm*) for the results of the asynchronous commands. In this event, you should define a new recipient, otherwise the results are lost. To do this, you can use the call KC_MODIFY_OBJECT with parameter type KC_MAX_PAR and the administration command KDCAPPL.

Deletion of program units, VORGANG exits, transaction codes and TAC queues has the following effects:

- Deleted program units and VORGANG exits can no longer be called.
- Asynchronous jobs to a deleted transaction code can no longer be created.
- Asynchronous jobs that are still in the message queue of a transaction code at the time of deletion are no longer processed. They do, however, remain in the message queue of the asynchronous TAC. To relieve capacity constraints in the page pool you should delete these asynchronous jobs from the queue (see KDCS call DADM in the openUTM manual „Programming Applications with KDCS”).
- No dialog services can be started to a deleted TAC. Dialog services that are open at the time of deletion can still be processed normally provided that only the service TAC is deleted. They are, however, terminated if a follow-up TAC is called which has already been deleted.
- When a TAC queue is deleted, its messages are deleted immediately. New messages cannot be created for a deleted TAC queue.

4.3.3 Deleting user IDs

You can remove a user ID from the configuration using either the “delayed” or the “immediate” delete method (see page 85). In UTM cluster applications only the delayed delete method is possible.

To delete a user ID from the configuration you must call KC_DELETE_OBJECT (with *subopcode1*=KC_DELAY or KC_IMMEDIATE) for the object type KC_USER.

Apart from the exceptions listed below, you can delete any user ID created explicitly in the configuration (statically or dynamically).

You cannot delete the following user IDs:

- KDCMSGUS, which openUTM creates internally for the MSGTAC service
- any user IDs assigned to a terminal for an automatic KDCSIGN (see page 76)
- all connection user IDs (i.e. user IDs that are permanently assigned to a client of the type UPIC, APPLI or socket)

In applications without explicitly generated user IDs, the deletion of user IDs created internally is generally not possible.

The following restrictions apply with regard to the point in time at which a user ID may be deleted:

You can only delete a user ID (delayed or immediate delete) if no user or client is signed on to the application at the time of deletion. For this reason, you should disable the user ID before deletion to avoid errors. Such disabling operations must occur in a separate transaction. To disable a user ID, see KDCUSER on page 797 or KC_MODIFY_OBJECT with *obj_type*=KC_USER on page 357.

Deleting a user ID is also temporarily not possible in the following cases:

- an asynchronous job is being processed, i.e. has been retrieved from the message queue and started.
- a distributed transaction is in PTC status (PTC = Prepare to Commit).
- the user-specific long-term storage area (ULS) of the user ID cannot be locked, e.g. because the administrator or an administration program is accessing it.

Delayed delete

Delayed deletion of a user ID has the following effects:

- No users/clients are able to sign on to the application with a user ID designated for a delayed delete.
- Asynchronous services which were started before the user ID was deleted and which are not being processed at the time of deletion are still able to run and can be administered. These services are not, however, able to create any more asynchronous jobs themselves.
- An open dialog service cannot be continued any further. Any service data that has been saved for a user (e.g. LSSB data, dialog messages) is deleted:
  - in the case of standalone applications, the next time the application is started
  - in UTM cluster applications, on the next start-up of the node application at which the user was last signed on

  The data is not deleted if an open service has a transaction in the PTC state. In this case, the transaction must first either be committed or rolled back. You can, for example, roll back transactions with the PTC state using the program interface (opcode KC_PTC_TA).
- ULS areas (ULS = user-specific long-term storage area) belonging to the user ID are still available for read and write accesses.
- All the messages in the message queue for this user ID are deleted immediately. No new messages can be created for this message queue.

Immediate delete

Immediate deletion of a user ID has the following effects:

- No users/clients are able to sign on to the application with an immediately deleted user ID.
- Asynchronous jobs which were generated and placed in the message queue by openUTM before the user ID was deleted, do not start, i.e. openUTM does not process them. They are deleted the moment openUTM retrieved them from the message queue for processing.

  If you query the information on jobs in the message queue using DADM RQ (see page 808), openUTM, instead of the user ID that issued the job, will output *NONE for the jobs of a deleted user ID.
- Jobs for LTERM or LPAP partners that are started before the user ID is deleted and are still in the partner's message queue, are sent.

- An open dialog service that was started by a deleted user ID, is also deleted immediately. There may be open dialog services for a user who is not signed on, e.g. if the user signed off during the service using KDCOFF after a synchronization point had already been reached.
- ULS areas (ULS = user-specific long-term storage area) belonging to the deleted user ID cannot be accessed. They are deleted.
- All the messages in the message queue for this user ID are deleted immediately.

4.3.4 Deleting key sets

Key sets can only be deleted from the configuration after a delay. To delete a key set, you have to call KC_DELETE_OBJECT (with *subopcode1*=KC_DELAY) for the object type KC_KSET.

Restriction: The KDCAPLKS key set cannot be deleted at all.

Objects that reference a deleted key set lose their access rights. However, other key sets can be assigned dynamically to TACs, TAC queues and user IDs.

4.3.5 Deleting LU6.1 connections and sessions

To delete an LU6.1 transport connection between the local UTM application and a partner application, you must call KC_DELETE_OBJECT (in standalone applications with *subopcode1*=KC_IMMEDIATE, in UTM cluster applications with KC_DELAY) for the object type KC_CON. If you want to delete an LU6.1 session, call KC_DELETE_OBJECT (with*subopcode1*=KC_DELAY) for the object type KC_LSES.

Deleting LU6.1 connections

It is not possible to delete a CON object when it is linked to the application.

Points to note when deleting LU6.1 sessions

An LSES object (half-session) can only be deleted when:

– The session is not set up, and
– Neither of the two half-sessions have the status PTC.

In order to check whether one of the half-sessions has the status PTC, you can query the status of the two half-sessions (e.g. by means of KC_GET_OBJECT with the object type LSES).

The following procedure is recommended for deleting an LSES object:

1. Set up the session before deleting the object.
2. Set the session to “quiet”.
3. Once the connection is set up, delete the object by means of the above call.

4.3.6 Deleting LTACs

Transaction codes by means of which service programs are started in partner applications can only be deleted from the configuration after a delay.

To delete an LTAC, you have to call KC_DELETE_OBJECT (with *subopcode1*=KC_DELAY) for the object type KC_LTAC.

4.4 Modifying object properties

You can use the KC_MODIFY_OBJECT call during an application run to modify the properties of objects and parameters of the application program and initiate actions (e.g. resetting of statistical values).

KC_MODIFY_OBJECT on page 311

The following object types have properties that can be modified dynamically:

KC_CLUSTER_NODE, KC_KSET, KC_LOAD_MODULE, KC_LPAP, KC_LSES, KC_LTAC, KC_LTERM, KC_MUX, KC_OSI_CON, KC_OSI_LPAP, KC_PTERM, KC_TAC, KC_TACCLASS, KC_TPOOL, KC_USER.

The following sections describe how to modify certain object types in more detail (KC_PTERM, KC_LTERM, KC_TAC, KC_USER, KC_KSET and KC_LSES).

The following parameter types have properties that can be modified dynamically:

KC_CLUSTER_CURR_PAR, KC_CLUSTER_PAR, KC_CURR_PAR, KC_DIAG_AND_ACCOUNT_PAR, KC_MAX_PAR, KC_TASKS_PAR, KC_TIMER_PAR.

You can modify a single object with each KC_MODIFY_OBJECT call. However, it is possible in an administration program to call KC_MODIFY_OBJECT more than once in order to modify the properties of multiple objects. In the call you specify the type of the object, its name and the properties to be modified.

When modifying application parameters, in a single call you can modify all the parameters that belong to the same parameter type.

The section entitled “KC_MODIFY_OBJECT - Modify object properties and application parameters” on page 311 explains which properties can be modified for which object type or application parameter and which actions are thus initiated.

The effectiveness and duration of a change depends on the object type or application parameter and on the property that is changed. Some changes apply only to the current application run, whereas others apply beyond it as well (durable). A change can take effect:

– immediately
– after transaction processing (PEND)
– when the utilization of the application permits it

UTM cluster applications

The following applies in a UTM cluster application:
Depending on the object, the call can initiate actions that apply either globally in the cluster or locally in the node. Actions with a global effect apply to all the node applications in the UTM cluster application irrespective of whether a node application is currently active or not. Actions with a local effect only apply to the node applications at which they are executed.

4.4.1 Modifying clients/printers and LTERM partners

In order to modify the properties of a client or printer, you have to call KC_MODIFY_OBJECT with the object type KC_PTERM. To modify the properties of an LTERM partner, you must specify the object type KC_LTERM.

LTERM partners that belong to an LTERM pool or clients/printers that are connected via an LTERM pool cannot be modified.

In the case of clients/printers and LTERM partners, you can change the status and the current state of the connection to the client/printer. A change of status (enabled/disabled) continues to apply after transaction processing beyond the end of the application run. A change to the current state (connection in existence, not in existence, currently being set up) applies when permitted by the utilization level of the application, but not after the end of the application run.

B B B B If the LTERM partner is assigned to a terminal, you can change the format attributes. However, a specific start format can only be used for applications without user IDs or when a separate sign-on service is defined. A change to the format attributes continues to apply after transaction processing beyond the end of the application run.

If you want to change the assignment of a client/printer to an LTERM partner, the partner must be in existence (it must not have been deleted). The LTERM partner must not be configured for connection to a client of the type UPIC. In addition, the LTERM partner must not be the master slave of an LTERM bundle or an alias or primary LTERM of an LTERM group. A change to the assignment continues to apply after transaction processing beyond the end of the application run.

In the case of clients/printers, only the LTERM partner, if assigned, or only one mode may be modified for automatic connection setup at the startup of the application. It is only possible to request automatic connection setup at startup of the application if the client/printer is not disabled. A change to connection setup at application startup continues to apply after transaction processing beyond the end of the application run.

4.4.2 Modifying transaction codes and TAC queues

In order to modify the properties of a TAC or a TAC queue, you must call KC_MODIFY_OBJECT with the object type KC_TAC.

It is not possible to change the status of a TAC and at the same time reset specific statistical values.

Changes to the status of a TAC or a TAC queue take effect immediately and continue to apply beyond the end of the application run. Changes to the statistical values of a transaction code take effect immediately.

If you want to control accesses to a transaction code by means of a key set, you can assign an existing key set to the access list of the transaction code. If there is a lock code, you have to remove it (set it to zero). Conversely, if access to the transaction code is protected by a lock code, there must not be a key set defined in the access list.

You can also protect a TAC queue against unauthorized reading/deletion and writing by means of a key set. To do this, assign the desired key set to the *q_read_acl* and/or *q_write_acl* parameters.

Changes to the parameters that control access continue to apply after transaction processing beyond the end of the application run.

Backup of messages in the dead letter queue in the event of processing errors can be enabled or disabled for asynchronous transaction codes using CALL=BOTH/FIRST and TAC queues. This backup option is not possible for MSGTAC and KDCDLETQ. Enabling and disabling of backup to the dead letter queue remains in effect after the end of the transaction and beyond the application run.

4.4.3 Modifying user IDs

In order to modify the properties of a user ID or the assigned USER queue, you have to call KC_MODIFY_OBJECT with the object type KC_USER.

You cannot disable user IDs with administration authorization, nor can you modify properties of user IDs that are assigned to a client of the type APPLI, SOCKET or UPIC.

If you want to change the password for a user ID, ensure that:

- The new password corresponds to the complexity level defined for the user ID.
- The existing password is not reused when it is only possible to use passwords with a limited period of validity for the user ID.

You can supply a user ID with access rights (key set) or change them.

You can use a key set to protect a USER queue against unauthorized reading/deletion and writing. To do this, assign the desired key set to the *q_read_acl* and/or *q_write_acl* parameters (see page 573).

Any changes you make to the properties of a user ID or a USER queue continue to apply after transaction processing beyond the end of the application run.

4.4.4 Modifying key sets

In order to modify the keys of a key set, you must call KC_MODIFY_OBJECT with the object type KC_KSET.

Note that the KDCAPLKS key set cannot be modified and that it is not permissible to specify a key less than 1 or greater than the maximum value permitted in the application (KDCDEF statement MAX, KEYVALUE operand).

Key sets with the MASTER attribute cannot be modified either.

4.4.5 Modifying LU6.1 sessions

In order to modify the properties of an LU6.1 session, you must call KC_MODIFY_OBJECT with the object type KC_LSES.

For an LU6.1 session you can initiate connection establishment or connection cleardown and, in the case of connection establishment, assign a transport connection to the session.

If you request the immediate establishment of a connection, the QUIET property must not be set and the LPAP partner must not be disabled. If you request the immediate cleardown of a connection, none of the other properties must be modified.

When specifying a transport connection for the session, you should ensure that the connection exists and is generated for the associated LPAP partner.

Any changes you make to an LSES object do not take effect unless the utilization level of the application permits it.

5 Generating KDCDEF statements from the KDCFILE

To ensure that regeneration does not cause you to lose the changes you made to your configuration while the application was running, openUTM provides you with the inverse KDCDEF. You can use this inverse KDCDEF to generate control statements for the UTM tool KDCDEF from current configuration data in the KDCFILE.

KDCDEF control statements generated by the inverse KDCDEF

The inverse KDCDEF generates control statements for the object types for which dynamic entry and deletion is possible. The inverse KDCDEF does not generate control statements for other objects and components in the application or for application parameters. However, you can use the inverse KDCDEF to generate the following KDCDEF control statements:

- USER statements

  For all user IDs that currently exist in the application. The inverse KDCDEF does not create any USER statements for the user IDs created internally by openUTM.

  In applications without user IDs, the inverse KDCDEF does not generate any USER statements.

- LTERM statements

  For all LTERM partners in the application which do not belong to an LTERM pool or a multiplex connection.

- PTERM statements

  For all clients and printers entered in the configuration. For clients belonging to an LTERM pool or a multiplex connection, no PTERM statements are generated.

- PROGRAM statements

  For all program units and exits currently contained in the configuration of that application.

- TAC statements

  For all transaction codes and TAC queues in the application.

- KSET statements

  For all the application's key sets.

- CON statements

  For all the application's LU6.1 connections.

- LSES statements

  For all the application's LU6.1 sessions.

- LTAC statements

  For all the transaction codes for partner applications.

The inverse KDCDEF generates control statements for all objects in the application belonging to one of these object types, regardless of whether the objects were entered in the configuration dynamically or were generated statically during a previous KDCDEF generation process. All modifications which you performed for this object during the application run are taken into account.

The inverse KDCDEF **does not** generate **any** control statements for objects which were deleted dynamically from the configuration of this application. After the next regeneration, these objects are therefore deleted completely from the configuration. They then cease to occupy any space in the table and the names of these objects can reused during regeneration.

Over and above this, after regeneration with KDCDEF, the UTM tool KDCUPD does not transmit any application data relating to the dynamically deleted objects from the old KDCFILE to the new KDCFILE, even if there is an object with the same name and object type as a deleted object in the new KDCDEF generation process. In particular, no asynchronous jobs generated by LTERM partners or user IDs which have subsequently been deleted are passed from KDCUPD.

The USER statements generated by the inverse KDCDEF do not contain any passwords. For user IDs generated with a password, the inverse KDCDEF generates USER control statements in this form:

```
USER name, PASS=*RANDOM,....
```

After a new KDCFILE has been generated, i.e. after the following KDCDEF run, you must pass the passwords for user IDs to the new KDCFILE using the UTM tool KDCUPD (see the openUTM manual "Generating Applications"). This is also possible in a UTM-F application.

In the case of UTM cluster applications, the passwords are present in the cluster user file and do no have to be transferred to a new KDCFILE using KDCUPD.

5.1 Starting the inverse KDCDEF

You can start the inverse KDCDEF “online” or “offline”. “Online” means that you start the inverse KDCDEF during the application is running. “Offline” means that you start the inverse KDCDEF after shutting down the application run.

In both cases, you can call the inverse KDCDEF in such a way that it produces KDCDEF control statements for all possible objects. However, you can also call the inverse KDCDEF in such a way that it only generates control statements for specified object types, which are grouped together in the object groups CON, DEVICE, KSET, LSES, LTAC, PROGRAM and USER.

You can request KDCDEF control statements for just one or more of these groups.

Starting inverse KDCDEF online

In order to start an inverse KDCDEF run online, you must generate your own application program which calls KC_CREATE_STATEMENTS.

KC_CREATE_STATEMENTS on page 246

The time at which the KDCDEF run actually starts depends on whether or not, when the KC_CREATE_STATEMENTS call is placed, another service in the application currently has write access to the configuration data in that application. Distinctions must be drawn between the following cases:

- At the time the KC_CREATE_STATEMENTS call is made, transactions may be running which modify the configuration data of the application or which change the passwords or locales.
  In this case, the KC_CREATE_STATEMENTS call will generate an asynchronous job. The inverse KDCDEF run is not started until these transactions have been completed. However, new transactions of this kind cannot be started until the inverse KDCDEF run has been completed, i.e. until the asynchronous job has been processed.

  The following also applies in UTM cluster applications:
  In all running node applications, an administration action which applies globally to the cluster results in a transaction which may delay the start of the inverse KDCDEF. Conversely, the execution of a global administration action at a running node may be delayed if an inverse KDCDEF is currently running there.

- At the time of the KC_CREATE_STATEMENTS call, **no** transactions are running which modify the configuration data, passwords or locales.
  In this case, the inverse KDCDEF run is started immediately (synchronously). The run will already have been terminated when control is returned to the program unit. In other words, by this time, all requested KDCDEF control statements have been generated and stored in files.

Note on UTM cluster applications:
It is not possible to start an online inverse KDCDEF as long as node applications with different generations are running in a UTM cluster application.

An inverse KDCDEF run is not subject to transaction management.

With the aid of the inverse KDCDEF executed online, you can make all preparations for regenerating your application parallel to the application run. This minimizes the amount of downtime incurred.

You can also start the inverse KDCDEF online using the administration tools WinAdmin and WebAdmin.

Starting the inverse KDCDEF offline

You can start the inverse KDCDEF offline, i.e. not during application runtime, by calling the UTM generation tool KDCDEF and submitting the control statement CREATE-CONTROL-STATEMENTS.

CREATE-CONTROL-STATEMENTS; see the openUTM manual “Generating Applications”

Files generated by the inverse KDCDEF can then be processed in the same KDCDEF run, or in a later one.

5.2 Result of the inverse KDCDEF run

The inverse KDCDEF either writes all control statements to one file or it writes the control statements for each group of objects to separate files.

B B On BS2000 systems, the control statements can also be written to an LMS library element instead of a file.

You can pass the files written by inverse KDCDEF as input to KDCDEF when the application is regenerated. To do so, you must enter the control statement OPTION DATA=*filename* for each of these files.

You can pass the files generated by inverse KDCDEF as input files direct to KDCDEF. However, you can also edit the files as well, i.e. you can modify them before the next KDCDEF run.

B B Whether or not LMS library elements can be modified depends on their type – only text-type elements can be modified.

You define the names of files generated by inverse KDCDEF when starting the inverse KDCDEF. If no file with this name exists, a new one is created automatically. If a file of this name does exist, you can define whether it should be overwritten or appended.

5.3 Inverse KDCDEF for version migrations

When migrating to a new version of openUTM, you must first generate the KDCDEF control statements in the previous version, i.e. you must start the inverse KDCDEF in the previous version. You can use the files this KDCDEF generates as input files for KDCDEF in the new version of openUTM.

5.4 Recommendations for regeneration of an application

When operating a UTM application, it may prove unavoidable to regenerate the application, i.e. to perform another KDCDEF run. Possible reasons can include:

- The maximum values defined during generation must be adapted.
- New objects may have to be generated for distributed processing via LU6.1 or OSI TP because the server network has to be extended for distributed processing.

  A KDCDEF run is only required for distributed processing via LU6.1 when new LPAP objects have to be inserted. Objects of the type CON, LSES and LTAC, on the other hand, can also be created by means of dynamic administration (provided enough table spaces have been kept free by means of the RESERVE statement).
- New load modules, shared objects or DLLs must be inserted in the application program.
- The table spaces reserved for dynamic entry of objects in the configuration are occupied. The tables must be extended or objects marked for deletion must be deleted now to create spare table spaces.

You can minimize the application downtime resulting from this type of regeneration. To do this, please note the following **recommendations**.

- When first generating your application, you should distribute the control statements for KDCDEF across several files before making them available to KDCDEF with OPTION DATA=. In particular, you should write the control statements USER, LTERM, PTERM, PROGRAM, TAC, CON, KSET, LSES and LTAC and TAC to separate files. When doing so, ensure that all statements relating to one specific group (see page 103) are written to one file. In this way, you can replace these files with files generated by an inverse KDCDEF if you regenerate the application at a later time.
- Before regenerating the application and before starting the inverse KDCDEF run, you should dynamically delete all objects no longer intended for the new configuration (KC_DELETE_OBJECT). Compared with manual deletion, dynamic deletion of related control statements from the input file has the following advantages for KDCDEF:
  - Manual deletion of KDCDEF statements from the KDCDEF input file is messy and prone to errors. Due account must be taken of relationships between the objects and, hence, between the KDCDEF statements during the manual deletion process. If any such relationships are overlooked, you must repeat the KDCDEF run. This only adds to the downtime.
  - You can automate the procedures involved in regeneration by calling the offline inverse KDCDEF followed by KDCUPD, see openUTM manual “Generating Applications”.

Over and above this, please note that under certain circumstances, when objects are being deleted manually, data stored in the KDCFILE and relating to the deleted objects can be passed to the new KDCFILE by KDCUPD, which is executed in conjunction with the following regeneration operation:

You wish to prevent KDCUPD from transferring the data from the old KDCFILE for a given file (e.g. because the "new" object has the same name and type but different properties). However, with KDCUPD you can only exclude the transfer of data for all objects of a given type, but not for a given object. You should therefore delete the object from the configuration dynamically. The object should be included again in the new generation.

In this case, KDCUPD does not transfer the data belonging to this object, as KDCUPD does not transfer the data of deleted objects.

For information on update generations in a UTM cluster application, see the corresponding subsection in the openUTM manual “Using openUTM Applications”.

Example

The new configuration should contain a transaction code with the name of an asynchronous transaction code which existed in the “old” configuration. However, the new transaction code calls a different service (i.e. it is assigned to a different program unit). A distinction must be made between the following cases:

- The properties of the "old" transaction code have been changed:
  In this case, if you enter TRANSFER ASYNTACS=YES, KDCUPD transfers the message queue of the “old” transaction code to the new KDCFILE together with the asynchronous jobs in the queue and assigns them to the “new” transaction code. Entering KDCUPD with TRANSFER ASYNTACS=NO ensures that none of the message queues for asynchronous transaction codes are transferred from the old KDCFILE to the new one.
- The old transaction code was dynamically deleted from the configuration. In the new configuration, it is included again:
  In this case, even if you enter TRANSFER ASYNTACS=YES, KDCUPD does not transfer the message queue for the old transaction code to the new KDCFILE because KDCUPD does not transfer any data from deleted objects.

The same applies to message queues for LTERM partners and USER queues of users.

6 Administration using commands

To enable you to use the administration commands of openUTM, the following requirements must first be fulfilled:

- The standard administration program KDCADM must have been generated (KDCDEF statement PROGRAM) or included in the configuration dynamically (administration program with KC_CREATE_OBJECT and *obj_type*=KC_PROGRAM).
- The administration commands which you want to use must have been generated as transaction codes (KDCDEF statement TAC) or included in the configuration dynamically (administration program with KC_CREATE_OBJECT and *obj_type*=KC_TAC).

For details of KDCDEF generation for commands and of the authorization level required for calling commands, see chapter “Access rights and data access control” on page 153f.

The openUTM command interface provides a dialog command and an asynchronous command for every KDCADM administration function. You can therefore terminate all actions (exception: shutting down the application run with KDCSHUT KILL), either in dialog or message queuing.

openUTM commands can be issued by users on a terminal, by client programs and by partner applications. However, in the first instance, they are intended for terminal input. For administration by client programs and other applications, the program interface to administration is far more suitable.

6.1 Administration in dialog

The dialog administration commands can be used by:

– users on terminals
– UPIC clients
– TS applications
– LU6.1 or OSI TP partner applications
– other dialog program units in the application

The user IDs under which a client is signed on must have administrator authorization.

Input of administration commands

A user on the terminal must enter the commands in line mode. Formatted entries are not accepted (exception: commands which have no operands).

The advantage of entering commands in line mode is that command processing does not take much time and administration tasks can also be performed in conjunction with other services.

B
B

In the UTM application on a BS2000 system, entries for administration commands will be rejected if an edit profile was used the last time that output was issued.

Output of results

openUTM returns the result of command processing to the job-submitting service. Output to the terminal also occurs in line mode.

If output to a terminal does not fit on one page of the screen, openUTM offers a continuation prompt on the last line of each screen display which can be used to continue output from the current position.

The chapter “Administration commands - KDCADM” on page 663 describes what the result message for each command looks like in the section describing the relevant commands.

Output after successful processing of an administration command does not necessarily mean that the action you requested has been completed successfully. With some commands, the message merely means that openUTM has initiated the action (e.g. to establish a connection, to exchange programs). The reason for this is that it takes an extended period for these actions to be carried out or that openUTM is not able to execute the action until a later time. You can find out whether the appropriate action was carried out successfully by submitting a KDCINF query at a later date. With some of these actions (e.g. program replacement), openUTM generates K messages after processing is complete which indicate to you whether or not the action was performed successfully. These messages are usually sent to the message destination SYSLOG; output takes place in standard form (SYSOUT/stderr).

6.2 Administration using message queuing

The asynchronous commands can be called by:

– terminal users
– TS applications
– LU6.1 or OSI TP partner applications
– other dialog or asynchronous program units in the application

The users/(OSI-)LPAPs that call the commands must have administration authorization.

When an asynchronous command is submitted, an asynchronous job is generated which openUTM adds to the message queue of the relevant administration TACs of KDCADM. The job is then executed independently of the job-submitting service or program unit.

The asynchronous commands make “programmed or automatic administration” possible. The data supplied by the standard administration program KDCADM can be passed to another program unit which analyzes the data and initiates appropriate actions (calling additional commands or transaction codes). The asynchronous commands can, for example, be called by event service MSGTAC which responds to certain events (UTM messages) when an administration command is called.

Submitting administration commands

At a terminal, asynchronous commands must be entered in line mode, as they are with administration in dialog mode. Partner applications pass commands together with operands to the application. The same operands are passed as in dialog mode. The asynchronous commands differ from dialog commands only in terms of their name.

A KDCS program unit calls an asynchronous command, either by submitting an FPUT NE call or, if the command is to be executed by a certain time, by submitting a DPUT NE call.

You supply the name of the asynchronous command (=transaction code) to the KDCS parameter field KCRN of the call. The message area for the call must contain the operand list of the administration command. You must pass every administration command in an FPUT or DPUT call.

You can send several calls relating to the same administration command and which are to be processed in one transaction as message sections. Every message section must contain an administration command (including the operands). The administration program KDCADM reads the message sections in a loop of FGET calls and processes them.

FPUT NT		DPUT NT	First call of the administration command, e.g. KDCLTRMA
FPUT NT	or	DPUT NT	Second call from KDCLTRMA
...		...	Further calls from KDCLTRMA
FPUT NE		DPUT NE	Last call from KDCLTRMA

The user ID under which the program unit is running must have administration privileges. The MSGTAC program unit always has administration privileges (see also the description of the MSGTAC program unit in the openUTM manual „Programming Applications with KDCS”).

Output of the result

After the job has been processed, openUTM informs you of the result via an asynchronous message. This message has the following format:

Header
1st line of result (= 1st line on screen, as for dialog output)
2nd line of result (= 2nd line on screen, as for dialog output)

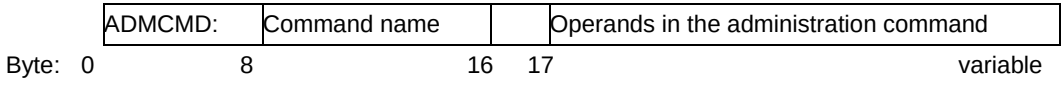
:

:

The result is output with the same number of lines as the corresponding dialog command. Only the line output in dialog mode for the scrolling function is omitted.

The structure of screen lines for dialog output is illustrated in chapter “Administration commands - KDCADM” on page 663 beside the description of the appropriate command.

Structure of header



Recipient for the result

All messages generated by the asynchronous commands go to the same recipient (DESTADM) which can be defined either during KDCDEF generation or at runtime by administration using either WinAdmin, WebAdmin or the KDCADMI program interface (*opcode*=KC_MODIFY_OBJECT and *object_type*=KC_MAX_PAR, see page 379). Administration can define a different recipient at any time. A recipient can take the form of an asynchronous TAC which further processes the result or the LTERM partner of a terminal, printer or a TS application.

If no recipient has been defined, openUTM still carries out the administration commands but the result messages are lost in the process.

However, if an asynchronous TAC is defined as the recipient, and if it is not available, e.g. because it is disabled, the command is not executed and openUTM generates the message K076.

If the recipient is an LTERM partner, the result is issued as an asynchronous message. If the recipient is an asynchronous TAC, the relevant program unit must read every single line of the result with an FGET call. The first FGET call supplies the header. Every subsequent call supplies one line of screen output.

The layout of the output is not subject to the compatibility guarantee, i.e. it may vary when changing to a new version of openUTM. Program units which evaluate the output from administration commands may therefore have to be adapted when a new version is installed.

Assignment of jobs to results for the recipient

When entering the operands of an asynchronous command, you can also enter a comment in inverted commas (“comment”). This comment can then be evaluated by the recipient for the results message.

As a comment you can, for example, enter a job number. The recipient can use this job number to identify the job.

In this case, the comment should be entered before the operand to ensure that job identification is always at the start of each message and is easy to address.

```
asynchronous command   "comment"  operands
```

7 Writing your own administration programs

The KDCADMI program interface allows you to write your own administration programs. You must always write an administration program as a KDCS program unit, i.e. it must be framed by an INIT and a PEND call. The PEND call should always terminate the transaction.

You can create administration programs:

– as dialog program units for administration in dialog mode
– as asynchronous program units for administration by means of message queues, e.g. for automatic administration, see chapter “Automatic administration” on page 147.

Every administration program has the following structure:

```
INIT
...
MGET (or FGET, if it is an asynchronous program)

... Analyse input

KDCADMI (call administration interface)
[KDCADMI] (several calls if necessary)

...

[RSET]

MPUT (or FPUT/DPUT)
PEND
```

You can submit several administration calls in an administration program. If you start a number of calls in a transaction, you must take account of the fact that some calls have to be made in a certain order and that a number of actions prompted by administration programs are subject to transaction management, i.e. they are not executed until a PEND call has been carried out successfully. In this case, you should provide a RSET call in the event of a fault.

A UTM application can have several administration programs for different purposes. An administration program can be started from a terminal, a client or another program unit (e.g. MSGTAC) or indeed from another application.

7.1 Dialog administration programs

If you wish to perform administration tasks in dialog mode, you can:

– group several administration jobs in one program, or
– program the administration tasks as a multi-step service and
B – input and output the data using formats

The two examples below outline how you can implement this.

7.1.1 Several administration calls

In this example, a load module, shared object or a DLL available in several versions is to be replaced at runtime with a new version and extended by a new program unit with a new TAC. The replacement operation runs in three steps.

First of all, a number of files must be requested by KDCADMI, e.g. the version of load module/shared object/DLL loaded that is before the configuration (TAC, PROGRAM statement) is modified in a second step. The actual replacement takes place in the final step.

```
#include <kcadminc.h>       /* Include file for the administration      */

INIT
...
MGET                         /* Read in data (name, TAC,...)             */
                             /* of prog. unit being replaced             */
... Analyse input

/********************* 1st section:check and query *********************/

KDCADMI opcode=KC_GET_OBJECT /* Is space for the TAC PROGRAM,...         */
                             /* statements reserved ?                    */

KDCADMI opcode=KC_GET_OBJECT /* Check whether TAC PROGRAM statements ... */
                             /* already exist                            */
KDCADMI opcode=KC_GET_OBJECT /* Determine current version of load module */
                             /* shared object                            */
if {error in section 1:
    MPUT with PEND FI }      /* If error message appears on screen       */
```

```
/*********************  2nd section: dyn. generation     *****************/
KDCADMI opcode=KC_CREATE_OBJECT
                             /* Insert PROGRAM statement                  */
KDCADMI opcode=KC_CREATE_OBJECT
                             /* Insert TAC statement                      */
if {error in section 2: RSET}/* roll back if fault in transaction         */
/********************* 3rd section: replacing program  ********************/

KDCADMI opcode=KC_MODIFY_OBJECT
                             /* Replace program unit                       */


MPUT                         /* Message on screen                         */
PEND FI
```

If errors occur in section 2, the RSET call is necessary to prevent inconsistent generation from occurring. The KC_CREATE_OBJECT operations must be specified for the objects shown in this sequence (PROGRAM TAC), otherwise openUTM is unable to generate the necessary references.

7.1.2 Multi-step service

In this example, information about the UTM application is retrieved in a first step and then, if necessary, object properties are modified in a second step. Both programs operate using a #format.

```
/************************ Program unit ADMREAD ***************************/

#include <kcadminc.h>       /* Header file for administration           */
INIT

MGET ... KCMF=#FORMADM       /* Entries are read in with a format        */
                             /* and the input is analyzed                */
KDCADMI opcode=KC_GET_OBJECT
                             /* Administration call, UTM sends data to   */
                             /* the program                              */
MPUT KCMF=#FORMADM           /* Output data/result to screen             */

PEND RE KCRN=ADMMOD          /* Service is continued                     */
```

```
/************************ Program unit ADMMOD ***************************/

#include <kcadminc.h>       /* Header file for administration           */
INIT
MGET ... KCMF=#FORMADM       /* Entries are read in with a format        */
                             /* and the input is analyzed                */
KDCADMI opcode=KC_MODIFY_OBJECT
                             /* The required object is modified          */
                             /* Several KDCADMI calls are possible       */
MPUT KCMF=#FORMADM           /* Output data/result to screen             */

PEND FI                      /* Service is terminated                    */
```

You can extend these programs, for instance, as follows:

– analyze the responses to the KDCADMI call and, in the event of errors, issue an appropriate message or
– write the data supplied to an LSSB in ADMREAD which can be reused in ADMMOD.

X/W X/W X/W openUTM on Unix, Linux and Windows systems does not support a formatting system, so if you want to call the program using *utmdtp* in a shell resp. DOS window, you must program the MGET and MPUT calls in line mode

You can also address this program using a UPIC client.

7.2 Diagnostic options for the administration interface

For error diagnosis for calls made to the administration interface, there are the two areas ADMI-DIAGAREA and ADMI-USERAREA in the UTM dump and the ADMI trace as a individual file. openUTM offers the following diagnosis options:

– In the UTM-DIAGAREA, the KDCS opcode ADMI displays the administration interface.

– A simultaneous log is kept for all calls in ADMI-DIAGAREA. The ADMI-DIAGAREA is structured in a similar manner to the UTM-DIAGAREA and is described cyclically.

– A simultaneous service-specific log is kept for all data transferred to openUTM in ADMI-USERAREA (data area or selection area). In each case, the ADMI-USERAREA only receives the data of one call to the administration interface.

– You can enable the ADMI trace to diagnose errors that occur in programs that use the administration program interface (KDCADMI).

B – On BS2000 systems, if SAT logging is activated and the UTM event ADM-CMD is selected, all calls to the administration interface are logged. In addition, in the case of opcode=KC_GET_OBJECT, the return codes KC_MC_OK and KC_MC_LAST_ELT are logged successfully.

For a description of ADMI-DIAGAREA, ADMI-USERAREA, ADMI trace and of the structure of the SAT log records, please refer to the relevant openUTM manual ”Messages, Debugging and Diagnostics” for the platform you are using.

8 Central administration of several applications

If you want to administer several UTM applications centrally, you can either use WinAdmin or WebAdmin or perform administration using your own command procedures or administration programs.

- **WinAdmin** and **WebAdmin** provide all the functions of the programming interface in a convenient user interface. You can administer several UTM applications running on different computers on BS2000, Unix, Linux or Windows systems at the same time.

  WinAdmin and WebAdmin are easy and quick to use, as no programming is required, either on the administration computer or in the UTM applications to be administered.

- You can create your own command procedures or programs if, for instance, you wish to use functions that are not provided by WinAdmin or WebAdmin.

  The administration tasks are split into a centralized part, the administration application, and a remote part which runs on the particular UTM application to be administered.

  You can handle central administration either via the command interface or via the program interface. You are advised to always use the program interface for the administration of the program interface.

  A number of basic models are available for configuring the central administration functions, see page 127.

Administration of UTM cluster applications

You can administer a UTM cluster application and the associated node applications together.

- **WinAdmin** and **WebAdmin** provide administration functions which you can apply globally to all of the node applications in the UTM cluster application. Furthermore, WinAdmin and WebAdmin allow you, for example, to display statistical summaries which include all the running node applications.

  For this reason, you are recommended to use WinAdmin or WebAdmin to administer UTM cluster applications.

- You can create your own command procedures or programs in the usual way. Additional data structures are available for administering UTM cluster applications:

  – The data structure *kc_cluster_par_str* is defined for the parameter type KC_CLUSTER_PAR. UTM uses *kc_cluster_par_str* to return the current settings for the global properties made in a UTM cluster application together with current data (e.g. generation time, start time, number of active and generated node applications) (see section “kc_cluster_par_str - Global properties of a UTM cluster application” on page 593).

  – The data structure *kc_cluster_node_str* is defined for the object type KC_CLUSTER_NODE. UTM uses *kc_cluster_node_str* to return the properties of the individual node applications (instances) in a UTM cluster application (see section “kc_cluster_node_str - Node applications of a UTM cluster application” on page 457.

  – The data structure *kc_cluster_curr_par_str* is defined for the object type KC_CLUSTER_CURR_PAR. UTM returns current values for the UTM cluster application in *kc_cluster_curr_par_str* (see section “kc_cluster_curr_par_str - Statistics values of a UTM cluster application” on page 592). In addition, *kc_cluster_curr_par_str* can be used to reset the statistics counters of the UTM cluster application.

  On page 131, you can find a generation example for the administration of a UTM cluster application via a UPIC client.

You can find further information on administering UTM cluster applications in the following manuals:

B – openUTM manual “Using openUTM Applications on BS2000 Systems”

X/W X/W – openUTM manual “Using openUTM Applications on Unix, Linux and Windows Systems”

8.1 Administration using WinAdmin and WebAdmin

This section provides you with an introduction to working with WinAdmin and WebAdmin. For detailed information, see

- the **WinAdmin Description** which provides a comprehensive overview of the range of functions and the WinAdmin handling. This document is supplied together with WinAdmin and is also available online as a PDF file.
- the WebAdmin description which provides a comprehensive overview of the range of functions and WebAdmin handling. This document is supplied together with WebAdmin and is also available online as a PDF file.
- the **online help system** which describes context-sensitively all the dialog boxes and associated parameters available in the graphical user interface of WinAdmin and WebAdmin. It also illustrates how to configure WinAdmin and WebAdmin in order to administer UTM applications.

WinAdmin and WebAdmin allow you to use the complete range of functions of KDCADMI, for instance to add objects to configurations dynamically, delete objects or start and terminate UTM applications. Furthermore, additional functions are available which cannot be accessed using KDCADMI:

– Definition of message collectors in order to query, display and archive UTM messages from the live UTM applications,
– Administration of message queues,
– Administration and control of printers,
– Reviewing the contents of GSSBs and deleting GSSBs,
– Creation and deletion of temporary queues,
– Grouping of several administration steps in a single transaction (only WinAdmin),
– Extremely comprehensive support for the UTM security concept using roles and access lists,
– Definition of actions such as storing statitic values in files or reacting to thresholds being exceeded or not met,
– Collection and archiving of statistical data on the UTM applications.

As far as openUTM is concerned, WinAdmin and WebAdmin area UPIC-R type clients. Before you can administer a UTM application using WinAdmin or WebAdmin, you must therefore

– generate WinAdmin or WebAdmin access in the UTM application (see page 124),
– and configure the connection parameters in WinAdmin or WebAdmin (see page 125).

8.1.1 Adapting generation of the UTM application

On the UTM application side, access to the program KDCWADMI and the UPIC connection from WinAdmin or WebAdmin must be generated.

Enabling access to the program interface

In order to enable access to the program interface, the program KDCWADMI and the TAC KDCWADMI must be generated. The following KDCDEF statements are required for this:

B `PROGRAM KDCWADMI, COMP=ILCS` BS2000 systems

X/W X/W `PROGRAM KDCWADMI, COMP=C` Unix, Linux and Windows systems

and `TAC KDCWADMI, PROGRAM=KDCWADMI, CALL=BOTH, ADMIN=Y`

The program unit KDCWADMI is supplied with openUTM and can be linked to the application or be dynamically loaded by the application.

openFT must be installed and configured if you want to use WinAdmin or WebAdmin to start UTM applications or use WinAdmin to initiate KDCDEF/KDCUPD runs. WinAdmin can send or retrieve data via FTP.

Making WinAdmin and WebAdmin known as a UPIC client

In addition, WinAdmin or WebAdmin must be generated as a UPIC client in all the openUTM applications to be administered using WinAdmin or WebAdmin. The following KDCDEF statements serve as an example (PTERM/LTERM):

B `BCAMAPPL` *bcamappl_name*`, T-PROT=RFC1006` (BS2000 system)

X/W X/W `BCAMAPPL` *bcamappl_name*`, T-PROT=RFC1006, LISTENER-PORT=`*port* (Unix, Linux and Windows system)

X/W Note: Although LISTENER-PORT is not a mandatory parameter, it is required in practice.

```
PTERM pterm-name, LTERM=lterm-name, BCAMAPPL=bcamappl-name,
                  PRONAM=processor-name, PTYPE=UPIC-R 1)
LTERM lterm-name
MAX PRIVILEGED-LTERM=lterm-name
USER wadmin, PASS=C'XYZ', PERMIT=ADMIN, RESTART=NO
```

1) WinAdmin or WebAdmin can also be generated via TPOOL instead of via PTERM/LTERM. However, it is then not possible to set up this connection as a privileged LTERM:

```
TPOOL LTERM=upiclt, NUMBER=10,PRONAM=*ANY,PTYPE=UPIC-R,
BCAMAPPL=bcamappl-name
```

The names *pterm-name*, *lterm-name*, *bcamappl-name, upiclt*, and *wadmin* are freely selectable in accordance with the naming conventions.
pterm-name is the name you give to the WinAdmin or WebAdmin client. *bcamappl-name* is the name you give to the application for client/server communication. *upiclt* is the prefix for the name of the LTERM partner, *wadmin* is an administration-authorized user ID for the application, and *XYZ* is the password for the *wadmin* user ID.

The assignment of a password is not mandatory, but a password should nevertheless always be used to maintain the security of the application.

You need the application name assigned here, the user ID and possibly the password in order to configure WinAdmin or WebAdmin.

8.1.2 Configuration of WinAdmin and WebAdmin

A configuration database is set up when WinAdmin and WebAdmin are started for the first time. The administration data of the UTM applications to be administered using WinAdmin or WebAdmin must be stored in this database to begin with. You use this data to specify the following on the WinAdmin and WebAdmin side:

– what the application is called
– the system on which the application runs
– the properties of the connection
– the users who can administer this application

This data is assigned to the WinAdmin or WebAdmin objects “Hosts”, “UTM Applications”, “UPIC Connections” and “WinAdmin Users“ or „WebAdmin Users”.

You can also define collections. A collection contains one or more UTM applications. By default, the collection <All UTM Applications> is set up.

When changing the WinAdmin or WebAdmin version, you can import the data of the previous version.

Configuration of WinAdmin and WebAdmin objects

The following table lists WinAdmin’s and WebAdmin‘s objects that have to be defined.

Object	Description and properties
Hosts	This object describes in WinAdmin or WebAdmin the system on which the UTM application runs (application host).
UTM Applications	This object describes the UTM application to be administered.
UPIC Connections	You use this object to define the connection from WinAdmin or WebAdmin to the application.

Object	Description and properties
WinAdmin / WebAdmin Users	After installation, only the WinAdmin/WebAdmin user ID “Master” is authorized to do everything. It is advisable to define further user IDs with restricted authorizations.
Collections	This object combines UTM applications to form a collection.

For details, see the description of WinAdmin and/or WebAdmin.

Working with collections

A WinAdmin/WebAdmin user can combine multiple applications to form a collection in order to simplify their administration.

Using WinAdmin, it is even possible to administer objects from different applications in an open collection together, i.e. in a single step.

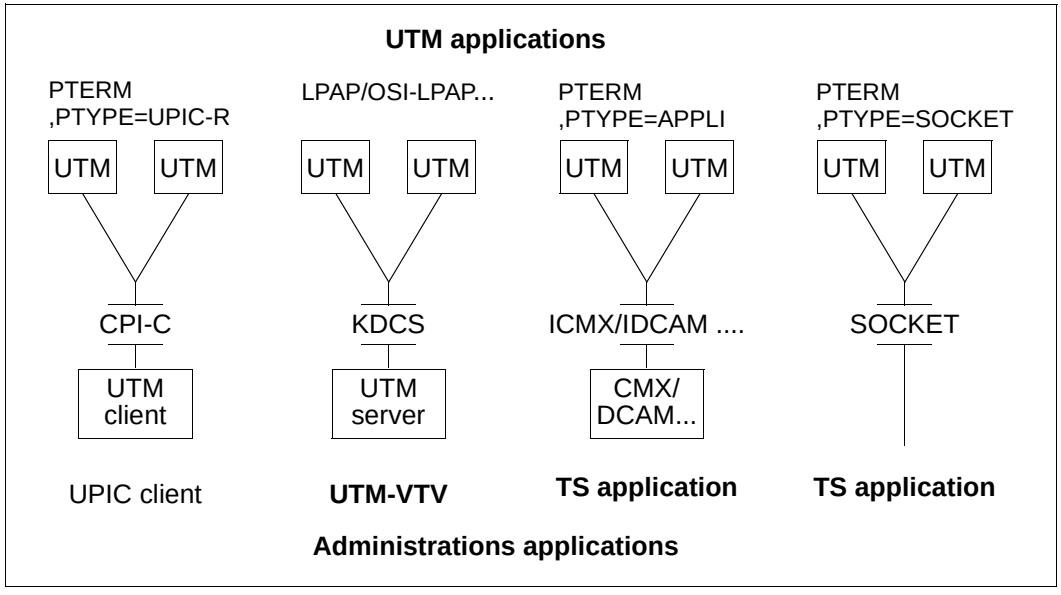
Checking availability

When you have performed the necessary configuration steps in UTM and WinAdmin/WebAdmin, you can check that the UTM application is accessible.

If the application is available, you can view its objects. These are displayed graphically in the WinAdmin/WebAdmin user interface in a tree structure or as a table.

8.2 Configuration models for own application of administration

You can implement the administration application as a UPIC client application, as a UTM application with distributed processing (with or without global transaction management) or as a TS application (SOCKET, CMX, DCAM, UTM...). The figure below illustrates all possibilities and the interfaces they use.



In all cases, the administration application must be generated with administration privileges in the applications to be administered.

The diagram applies equivalently for the administration of UTM cluster applications, see also “Generation example for a UTM cluster application” on page 131.

8.2.1 Administration via UPIC clients

A UPIC client can run on BS2000, Unix, Linux and Windows systems. If the platform you select is Windows system, you have the advantage of being able to generate a friendly graphical user interface for the administration program.

A client can also be restarted in that it can request the latest output message and continue the interrupted service; see the manual „openUTM-Client for the UPIC Carrier System”.

Please note that a UPIC client

- can only communicate with one application at any one time, if it is running under a BS2000 system
- cannot itself send any asynchronous jobs to openUTM
- always has to take the initiative, i.e. it cannot be started from the application to be administered.

UPIC clients for Unix, Linux and Windows systems are available for the products WinAdmin and WebAdmin.
WinAdmin and WebAdmin offer the full function scope of the KDCADMI program interface (see the section “Administration using WinAdmin and WebAdmin” on page 123).

UTM on BS2000 systems is supplied with a UPIC client program complete with an SDF command interface in the form of a fully compiled object code. You can adapt the configuration for this program to the needs of your own configuration. For more details, see section “The C program units ADJTCLT” on page 847 of the appendix.

Programming

What you program is a UPIC program which sends the data required for administration (the administration command or input for the administration command) to the remote application and receives the corresponding output from the application being administered. The diagram below gives a rough outline of a UPIC program for Unix, Linux or Windows systems.

```
#include <upic.h>

Enable_UTM_UPIC                          /* Sign on to UPIC carrier system  */
Initialize_Conversation                  /* Initialize conversation;        */
                                         /* sym_dest_name addresses the     */
                                         /* application to be administered. */
Set_TP_Name                              /* TAC for administration program  */
                                         /* or KDC.... administration TAC.  */

Set_Conversation_Security_Type=CM_SECURITY_PROGRAM
                                         /* Use UTM user concept            */
Set_Conversation_Security_User_ID        /* Set UTM user ID which must have */
                                         /* administration privileges.      */
Set_Conversation_Security_Password       /* Password for user ID            */
...
Allocate                                 /* Set up conversation.            */

memcpy (buffer,  )                       /* Supply data area with           */
                                         /* command or program input        */

Send_Data                                /* Send command/program input to   */
                                         /* the administered application.   */

Receive                                  /* Message returned by UTM appli-  */
                                         /* cation and then evaluated by    */
                                         /* the program.                    */

Disable_UTM_UPIC                         /* Sign off UPIC carrier system    */
```

How the UPIC program can send and receive data is described in section “Central Administration using commands” on page 140 and in section “Central Administration using programs” on page 141.

Generation example (standalone UTM application)

The UPIC program on a Unix or Linux system computer *UNIX0001* is to administer three UTM applications. One application is running on a BS2000 system computer *D123ZE45*, the second on a Unix computer *D234S012* and the third on a Windows computer *WSERV01*. The UTM applications are to be able to shut down with the administration TAC KDCSHUT and to call the administration program with the TAC TPADMIN.

1. Entries in the UPIC client's upicfile

   upicfile:

```
* Local name of the CPI-C application
LNADMIN001 UPIC0001;
* UTM application on a BS2000 system
HDUTMAW001 APPLIBS2.D123ZE45 TPADMIN;
* UTM application on a Unix or Linux system
SDUTMAW002 APPLUnix.D234S012 TPADMIN PORT=30000;
* UTM application on a Windows system
SDUTMAW003 APPLIWIN.WSERV01 TPADMIN PORT=30000;
```

2. UTM generation on the BS2000 system:

```
BCAMAPPL APPLIBS2,T-PROT=ISO
PTERM UPIC0001,PTYPE=UPIC-R,LTERM=UPICLTRM,
      ,BCAMAPPL=APPLIBS2,PRONAM=UNIX0001,...
LTERM UPICLTRM,KSET=ALLKEYS,USER=REMADMIN,RESTART=N
USER REMADMIN,PERMIT=ADMIN,RESTART=NO                  *)
TAC KDCSHUT, PROGRAM=KDCADM,ADMIN=Y                  **)
TAC TPADMIN,PROGRAM=ADMINPRG,ADMIN=Y,...
PROGRAM ADMINPRG,...
PROGRAM KDCADM
```

   The processor name *UNIX0001* must be generated in BCAM (by means of a BCIN or CREATE-PROCESSOR command or in the RDF). BCMAP entries are not required for RFC1006 via port 102.

3. UTM generation on Unix and Linux systems:

```
BCAMAPPL APPLUnix,LISTENER-PORT=30000,TSEL-FORMAT=T,T-PROT=RFC1006
PTERM UPIC0001,PRONAM=UNIX0001,TSEL-FORMAT=T,PTYPE=UPIC-R,LTERM=UPICLTRM,
      ,BCAMAPPL=APPLUnix
LTERM UPICLTRM,KSET=ALLKEYS,USER=REMADMIN,RESTART=N
USER REMADMIN,PERMIT=ADMIN,RESTART=NO                  *)
TAC KDCSHUT, PROGRAM=KDCADM,ADMIN=Y                  **)
TAC TPADMIN,PROGRAM=ADMINPRG,ADMIN=Y,...
PROGRAM ADMINPRG,...
PROGRAM KDCADM
```

4. UTM generation on WIndows systems:

```
BCAMAPPL APPLIWIN,LISTENER-PORT=30000,TSEL-FORMAT=T,T-PROT=RFC1006
PTERM UPIC0001,PRONAM=UNIX0001,TSEL-FORMAT=T
      PTYPE=UPIC-R,LTERM=UPICLTRM,BCAMAPPL=APPLIWIN
LTERM UPICLTRM,KSET=ALLKEYS,USER=REMADMIN,RESTART=N
USER REMADMIN,PERMIT=ADMIN,RESTART=NO                *)
TAC KDCSHUT, PROGRAM=KDCADM,ADMIN=Y                 **)
TAC TPADMIN,PROGRAM=ADMINPRG,ADMIN=Y,...
PROGRAM ADMINPRG,...
PROGRAM KDCADM
```

*) The connection user ID is used here, for which no password protection applies. If you require greater security, the UPIC client has to pass on a “genuine” user ID to openUTM using the CPI-C calls *Set_Conversation_Security_Type/_User_ID/_Password*. In this case the user ID must have administrator privileges and be password protected.

**) You should generate all the relevant TACs. KDCSHUT must always be generated. In the UPIC, program, the TAC can be set via the program (the default is TPADMIN).

Generation example for a UTM cluster application

The UPIC program on Unix or Linux system *UNIX0002* is to administer a UTM cluster application on the BS2000 systems *C123DE10*, *C123DE11* and *C123DE12*. The UTM cluster application *APPLBS2C* consists of three nodes and the administration program should be able to call it by means of the TAC REMADMIN.

1. Entries in the UPIC client’s upicfile:

   The UPIC client is configured in a way that requires a separate Symbolic Destination Name to be specified for each node.

```
* Local name of the CPI-C application
LNADMIN001 UPIC0001;
* UTM cluster application on the BS2000 system
HDclnode01 APPLBS2C.C123DE10  REMADMIN
HDclnode02 APPLBS2C.C123DE11  REMADMIN
HDclnode03 APPLBS2C.C123DE12  REMADMIN
```

   In this case, the UPIC program must explicitly address the relevant node (*clnode01 , clnode02* or *clnode03*).

2. UTM generation on the BS2000 system (initial KDCFILE):

```
BCAMAPPL APPLBS2C,T-PROT=ISO
PTERM UPIC0001,PTYPE=UPIC-R,LTERM=UPICLTRM,
      ,BCAMAPPL=APPLBS2C,PRONAM=UNIX0002,...
LTERM UPICLTRM,KSET=ALLKEYS,USER=REMADMIN,RESTART=N
USER ADMUSR01,PERMIT=ADMIN,RESTART=NO                    *)
USER ADMUSR02,PERMIT=ADMIN,RESTART=NO                    *)
USER ADMUSR03,PERMIT=ADMIN,RESTART=NO                    *)
TAC KDCSHUT, PROGRAM=KDCADM,ADMIN=Y                  **)
TAC REMADMIN,PROGRAM=ADMINPRG,ADMIN=Y,...
PROGRAM ADMINPRG,...
PROGRAM KDCADM
...
CLUSTER ...
CLUSTER-NODE ...,HOSTNAME=C123DE10
CLUSTER-NODE ...,HOSTNAME=C123DE11
CLUSTER-NODE ...,HOSTNAME=C123DE12
```

*) For each node, you should generate a user ID with administration authorizations since, by default, a user in a UTM cluster application continues to be signed on when the conversation terminates. The UPIC program must assign the user ID.

**) You should generate all the relevant administration TACs. In the UPIC, program, the TAC can be set via the program (the default is REMADMIN).

The processor name *UNIX0002* must be generated in BCAM (by means of a BCIN or CREATE-PROCESSOR command or in the RDF). BCMAP entries are not required for RFC1006 via port 102.

8.2.2 Administration via distributed processing

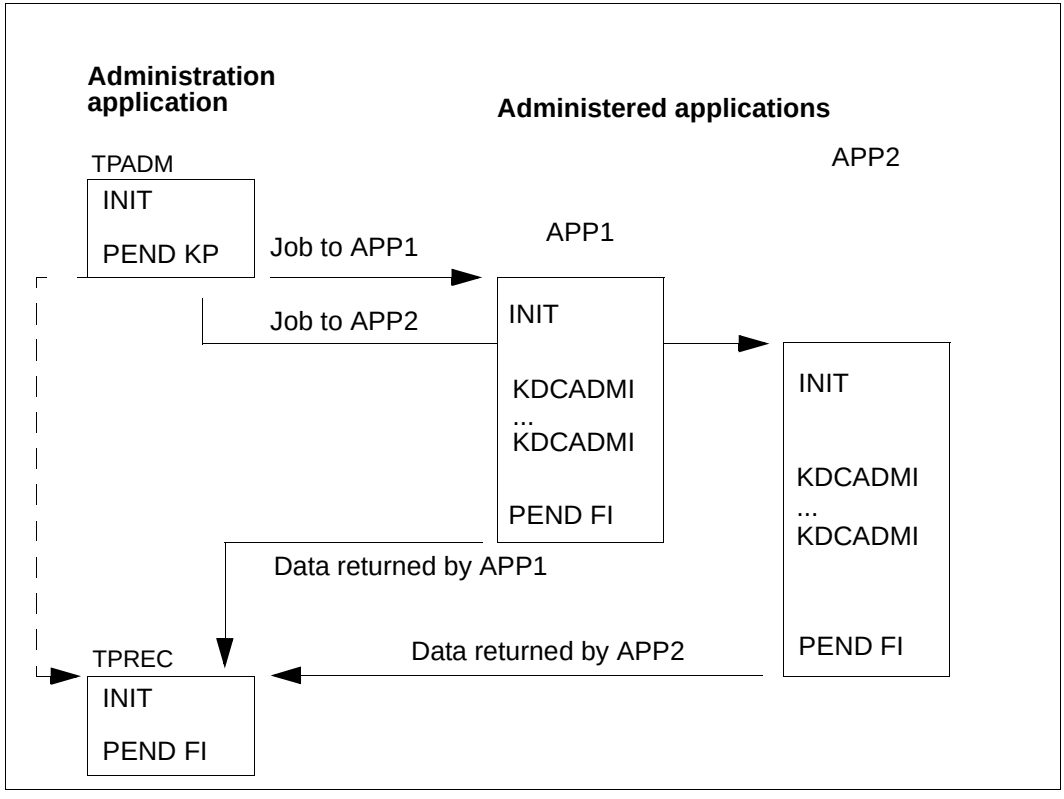
If you want to handle central administration for openUTM via distributed processing, you have the following advantages:

- Several applications can be administered simultaneously.
- Administration jobs can be started both from the administration application itself and from the applications being administered (the polling function).
- Time-driven administration jobs can be set up very easily (DPUT).
- You can, if necessary, work with global transaction management. This allows you, for example, to ensure that certain application parameters are modified simultaneously for all applications, which cannot be guaranteed when administering applications via a UPIC client or a TS application (as network failures can mean that the operation cannot be performed for one of the applications while the others are already working with the new values).

You can use the LU6.1 or OSI TP protocols for communication between the administration application and the servers being administered.

Programming

If you require global transaction management for your administration operations, one transaction from the administration application will need to communicate with several job receivers. The figure below illustrates this principle using the example of two administered applications, each of which submits several administration calls.



The program TPADM sends jobs to both applications. The program TPREC is called only after responses have been received from both applications. Once both applications have completed their respective jobs properly, TPREC terminates the global transaction and the service.

The following example gives an idea of what the programs TPADM and TPREC might look like. The administrative task is, from a Unix computer, to initiate the simultaneous replacement of a program in a UTM application on a Unix or Linux system and a UTM application on a BS2000 system. Program replacement is handled differently on Unix, Linux and Windows systems and BS2000 systems, however. BS2000 systems determine the current version of the load module, marks the load module for replacement and then reloads the application. On Unix, Linux and Windows systems, the program is replaced immediately. The administered applications can use a program like the one on page 116. The figure below illustrates this example for LU6.1 and OSI TP without global transaction management.

If you are using a UTM application on Windows systems, either instead of the administered or the administering UTM application on Unix or Linux systems, or both, then programming and generation are the same. Note that port number 102 cannot be used for UTM applications on Unix, Linux and Windows systems.

```
/* Program unit TPADM sends data to applications UTMAPPL1 and UTMAPPL2    */

INIT
memcpy (buffer,  ...)                  /* Edit data.                       */
APRO DM KCPI=VGID1 KCPA=UTMAPPL1       /* Address job-receiving service   */
        KCRN=TPADMIN                   /* TPADMIN in UTMAPPL1.            */

MPUT NE buffer                         /* Send data to UTMAPPL1.          */
     KCRN=VGID1

APRO DM KCPI=VGID2 KCPA=UTMAPPL2       /* Address job-receiving service   */
        KCRN=TPADMIN                   /* TPADMIN in UTMAPPL2.            */

MPUT NE buffer                         /* Send data to UTMAPPL2.          */
     KCRN=VGID2

PEND KP KCRN=TPREC                     /* Wait for job receiver.          */
```

For OSI TP with global transaction management, additional statements are required in order to:

– select the commit functional unit
(APRO... KCOF=C)

– request UTMAPPL1 to initiate the end of the transaction and dialog
(CTRL PE, KCRN=VGID1)

– request UTMAPPL2 to initiate the end of the transaction and dialog (CTRL PE, KCRN=VGID1)

```
/* Follow-up program TPREC receives confirmation from job-receiving       */
/* service                                                                */

INIT
   KCRPI=VGIDx                          /* 1st message comes from JS       */
                                        /* service with service ID VGIDx.  */
MGET NT KCRN=VGIDx                      /* Read response from JS service 1,*/
        KCRCCC=12Z KCRPI=VGIDy          /* Further message from other JS   */
                                        /* service (VGIDy) already waiting */

if (OK)                                 /* JS service 1 has initiated      */
 {                                      /* program replacement.            */
   MGET NT KCRN=VGIDy                   /* Read response from JS service 2.*/
           KCRCCC=1OZ KCRPI=SPACES      /* No further messages waiting.    */

  if (OK)                               /* JS service 2 has initiated      */
    {                                   /* program replacement.            */
      MPUT NE                           /* Send message to administrator.  */
      PEND FI                           /* Terminate global transaction.   */
    } else error_routine();
 } else error_routine();
....
error_routine ()                        /* Error routine                   */
        { MPUT NE                       /* Notify administrator            */
          PEND FR }                     /* roll back and terminate         */
                                        /* global transaction.             */
```

Generation example

The example shows an LU6.1 generation; the administration application uses two-level addressing.

In the example the port numbers and computer names (BS20HOST, UnixHOST, UnixADMI) are specified in the generation statements. See the openUTM manual “Generating Applications” under "Providing address information" for further information.

1. Generation of the UTM administration application on Unix or Linux systems

```
BCAMAPPL ADMINAPP,LISTENER-PORT=1234,T-PROT=RFC1006,T-SEL-FORMAT=T
***
*** Connection to application on Unix or Linux system; the administrator
*** application is the job submitter.
SESCHA ADMAPPL1,PLU=Y,CONNECT=Y
LPAP UTMAPPL1,SESCHA=ADMAPPL1
LSES ADMAG1,LPAP=UTMAPPL1,...
```

```
CON APPLUnix,BCAMAPPL=ADMINAPP,PRONAM=UnixHOST -
               ,LISTENER-PORT=2345,LPAP=UTMAPPL1,...
***
*** Connection to application on the BS2000 system;
*** the administrator application is the job submitter.
SESCHA ADMAPPL2,PLU=Y,CONNECT=Y
LPAP UTMAPPL2,SESCHA=ADMAPPL2
LSES ADMAG2,LPAP=UTMAPPL2,...
CON APPLIBS2,BCAMAPPL=ADMINAPP,PRONAM=BS2OHOST -
               ,LISTENER-PORT=102,LPAP=UTMAPPL2,...
***
*** LTAC for the remote administration program; two-level addressing
*** LTAC=RTAC is the TAC in the remote application.
LTAC TPADMIN
***
*** TACs for both administration programs
TAC TPADM,PROGRAM=...
TAC TPREC,PROGRAM=...
```

2. Generation of the administered UTM application on the BS2000 system

```
BCAMAPPL APPLIBS2,T-PROT=ISO
***
*** LU6 generation for the job receiver
SESCHA ADMINREC,PLU=N,CONNECT=N
LPAP UTMADMIN,SESCHA=ADMINREC,PERMIT=ADMIN
LSES ADMAN,LPAP=UTMADMIN,...
CON ADMINAPP,BCAMAPPL=APPLIBS2,PRONAM=UnixADMI,LPAP=UTMADMIN,...
***
TAC TPADMIN,PROGRAM=ADMINPRG,ADMIN=Y
PROGRAM ADMINPRG,...
```

3. Generation of the administered UTM application on Unix or Linux systems

```
BCAMAPPL APPLUnix,LISTENER-PORT=1234,T-PROT=RFC1006,T-SEL-FORMAT=T
***
*** LU6 generation for the job receiver
SESCHA ADMINREC,PLU=N,CONNECT=N
LPAP UTMADMIN,SESCHA=ADMINREC,PERMIT=ADMIN
LSES ADMAN,LPAP=UTMADMIN,...
CON ADMINAPP,BCAMAPPL=APPLUnix,PRONAM=UnixADMI -
               ,LISTENER-PORT=2345,LPAP=UTMADMIN,...
***
TAC TPADMIN,PROGRAM=ADMINPRG,ADMIN=Y
PROGRAM ADMINPRG,...
```

8.2.3 Administration via a TS application

The application can be any TS application such as a CMX application (PTYPE=APPLI) or a socket application (PTYPE=SOCKET), for example. However, you can also use a UTM application, which you generate as a TS application. The administration application is linked to the administered UTM applications by means of an LTERM/PTERM or TPOOL statement.

In all cases, the application can:

- simultaneously administer several UTM applications
- be started by the administered applications

How the application can be programmed depends on the type of TS application used. If you are using a UTM application, you can also use DPUT to send time-driven jobs to the administered applications.

In order to carry out administration by means of a TS application, one of the following cases must apply:

- The connection user ID must have administration authorization, e.g.:

  ```
  LTERM ADMINLTM,KSET=ALLKEYS,RESTART=N, USER=ADMINUS
  USER ADMINUS, PERMIT=ADMIN, RESTART=N
  ```

  or

- A genuine user ID with administration authorization must be signed on during the sign-on process for the TS application.

Generation

For the generation of an administered UTM application on a BS2000 system, it should be possible to call the command KDCSHUT and to call the administration program with the TAC TPADMIN.

To achieve this, the following statements will be required in the decentralized application for LTERM, TAC and PROGRAM, irrespective of whether the central application is a socket, CMX or DCAM application:

```
*********************************************************************
*** LTERM, TAC and PROGRAM
*********************************************************************
LTERM ADMINLTM,KSET=ALLKEYS,RESTART=N
USER ADMINLTM, PERMIT=ADMIN,RESTART=N
TAC KDCSHUT, PROGRAM=KDCADM,ADMIN=Y
TAC TPADMIN,PROGRAM=ADMINPRG,ADMIN=Y,...
PROGRAM ADMINPRG,...
PROGRAM KDCADM
```

To address the central application you must write the following statements depending on which type of application (DCAM or CMX) you are using. If you are using a UTM application, the same applies depending on whether the application is linked via NEA or via TCP/IP.

```
*********************************************************************
*** DCAM application which communicates via
*** NEA protocols with openUTM applications: on BS2000 systems
*********************************************************************
BCAMAPPL APPLIBS2,T-PROT=NEA
PTERM dcam-name,PTYPE=APPLI,LTERM=ADMINLTM,
BCAMAPPL=APPLIBS2,PRONAM=dcam-computer
*********************************************************************
*** CMX application on Unix or Linux system via TCP/IP-RFC1006
*********************************************************************
BCAMAPPL APPLUnix,T-PROT=RFC1006
PTERM t-selector,PTYPE=APPLI,LTERM=ADMINLTM,BCAMAPPL=APPLUnix,
        LISTENER-PORT=port-number, PRONAM=unix-computer
*********************************************************************
*** Socket application on Unix or Linux system
*********************************************************************
BCAMAPPL SOCKETBS,LISTENER-PORT=12000,T-PROT=SOCKET
PTERM SOCKPTRM,PTYPE=SOCKET,LTERM=ADMINLTM, BCAMAPPL=SOCKETBS,
               LISTENER-PORT=port-number, PRONAM=unix-computer
```

dcam-name and *dcam-computer* are the respective names of the DCAM application and computer on which the DCAM application is running. *t-selector* is the T selector for the remote CMX application. *unix-computer* is the name of the computer on which the CMX or socket application runs. *port-number* is the port number at which the central CMX or socket application waits for connection setup requests.

8.3 Central Administration using commands

Alongside the program interface, openUTM also provides the command interface for administration. However, the command interface only provides a subset of the functionality available in the program interface.

You can use both synchronous and asynchronous commands for central administration. In either case, the central administration program will have to:

– make the command available in the prescribed syntax
– send it to the administered UTM application in the form of a message.

The application being administered executes the command as if it had issued it itself. To be able to evaluate the command output in the central application you will, however, need to observe the differences inherent in synchronous and asynchronous methods.

Synchronous commands

If you use synchronous administration commands for central administration, the command output will be returned automatically to the sender, i.e. to the administration program.

This means that any configuration model is suitable for central administration with synchronous commands. If you are using a UPIC client for Windows systems, you can, for example, write a Visual Basic program to allow you to enter the administration commands via a friendly Windows interface. The program is able to filter openUTM's response before issuing any output so that you only see the parameters that are of importance to you. You can then implement the message interface to openUTM via a CPI-C program as described in section "Administration via UPIC clients" on page 128f.

Asynchronous commands

If you use asynchronous administration commands for central administration, the output is not returned automatically to the sender. The destination for command output must therefore be generated with MAX DESTADM in the decentralized applications.

If the central application is a TS application, then specify the LTERM name for the central application in MAX DESTADM. However, please note that the central application receives this output asynchronously, i.e. it has to determine the sender.
If you want to handle administration operations in the context of distributed processing, you must also use MAX DESTADM=TAC to add a further decentralized asynchronous program which receives the output and forwards it with FPUT to the administration application.

8.4 Central Administration using programs

If you are using the program interface, you can split the tasks in one of two ways between the administration application and the applications to be administered:

- Decentralized administration programs:
  You can use the program interface in such a way that a complete administration program exists within the administered application which can autonomously determine the necessary parameters and evaluate the data returned to it.
- Central administration programs
  You can use the program interface in the administered application purely as a message interface, i.e. it receives all parameters from the administration application and returns the results of the call (return codes, data) without verification.

8.4.1 Decentralized administration programs

If the administered applications use complete administration programs as described in chapter “Writing your own administration programs” on page 115, the control of an administration service will essentially reside with the application that is being administered. The administration program must therefore:

- interpret a message received from the administration application or - in the case of automatic administration, for example - from an application-internal MSGTAC program
- correctly supply all areas for the administration call
- evaluate and respond to the return codes, i.e. it must notify the administration application in the event of errors and, where appropriate, roll back the transaction
- evaluate the returned data and decide what data is to be sent to the administration application.

It is advisable to write individual program units for the various administration tasks or, if you are using a complete administration program, to address the program with different TACs depending on the task required. This will ensure that the tasks is selected on the basis of the TAC and not on the basis of the message.

Portable administration programs

If you want to use your administration programs in different applications running on different platforms, you can write the relevant programs in such a way that they can run both on Unix, Linux or Windows systems and BS2000 systems.

This task is simplified by the fact that the program interface has the same data structures on all platforms. You will, however, need to note the following platform-specific differences:

- There are certain fields and substructures which only have any meaning on one platform
  - When reading data, fields which are not relevant to the given platform are always populated with binary zeros.
  - When modifying or generating objects, the fields which are not relevant to the given platform must be populated with binary zeros. For this reason, the program should first establish the platform on which it is running. To do this it has to evaluate the field *system_type* in the structure *kc_system_par_str* after calling KDCADMI with the following parameters:

    opcode=KC_GET_OBJECT
    subcode1=KC_APPLICATION_PAR
    obj_type=KC_SYSTEM_PAR

    Once it has determined which platform it is running on, the program must first reserve the fields that are valid for all of the operating systems for the administration calls themselves. It then reserves the fields that are needed for the relevant platform.
- The sort order for characters differs between BS2000 systems and Unix, Linux and Windows systems: BS2000 systems generally use an EBCDIC code and Unix, Linux and Windows systems an ISO code.
- Names on BS2000 applications only use uppercase letters, whereas Unix, Linux and Windows systems names can use both lowercase and uppercase.
- Under certain circumstances, bytes can be represented differently on Unix, Linux or Windows systems and BS2000 systems (the little/big endian problem).

The following example shows a portable administration program which replaces a load module, shared object or DLL in the decentralized application. The program verifies which platform it is running on and uses the result to effect a program-internal branch.

On Unix, Linux and Windows systems, only the shared object/DLL is replaced, whereas BS2000 systems check whether the load module is in a common memory pool and, therefore, whether the application in fact needs to be replaced.

```
#include <kcadminc.h>       /* Include file for the administration      */

INIT
...
MGET                         /* Read in name/date of the program unit    */

... Analyze input

KDCADMI opcode=KC_GET_OBJECT /* Query operating system                   */


KDCADMI opcode=KC_GET_OBJECT
                             /* Determine current version of load        */
                             /* module and check whether it is at all    */
                             /* possible to replace it.                  */


if (BS2000)                  /* BS2000 routine                           */
   { KDCADMI opcode=KC_GET_OBJECT
                             /* Query load mode and determine whether    */
                             /* program is marked for replacement.       */
     KDCADMI opcode=KC_MODIFY_OBJECT
                             /* Replace or mark load module if it is     */
                             /* in a common memory pool.                 */
     if (common memory pool)
         KDCADMI opcode=KC_CHANGE_APPLICATION
                             /* Replace application                      */
   }                         /* End of the BS2000 routine                */

else                         /* Unix/Linux/Windows routine */
     KDCADMI opcode=KC_MODIFY_OBJECT
                             /* Replace shared object/DLL             */
                             /* End of the Unix/Linux/Windows routine */

MPUT                         /* Message to the initiator                 */
PEND FI
```

The program can also be supplemented by means of dynamic generation (TAC, PROGRAM,...) as described in the example on page 116.

8.4.2 Central administration programs

You can use the program interface on the side of the applications to be administered as a dedicated message interface. In this case, control of the administration functions lies entirely with the administration application. This application supplies the four areas needed for each administration call with the data they require and uses MPUT NT/NE to send it to the administered application.

The administered application merely converts the data supplied to the syntax required by the administration interface and then calls it. This means that it checks neither the data supplied with MGET nor the codes and data returned by the call. The diagram below outlines a program of this type.

```
/*************************************************************************/
/*         Dialog program for the administered application               */
/*                                                                       */
/* The program has four buffers in which data is received:               */
/* parameter_area, identification_area, selection_area, data_area        */
/*************************************************************************/

INIT

MGET NT in parameter_area           /* Fully supplied parameter area     */
                                    /* for the administration interface  */
MGET NT in identification_area      /* The identification area is        */
                                    /* supplied as a function of the     */
                                    /* opcode for the parameter area.    */
MGET NT in selection_area           /* The data supplied to the selection */
                                    /* area depends on the operation and  */
                                    /* may only have the length 0.       */
MGET NE in data_area                /* Data is supplied where necessary;  */
                                    /* otherwise the length 0 is supplied */

KDCADMI (&parameter_area,           /* The program calls KDCADMI without  */
         &identification_area,      /* checking the data.                 */
         &selection_area,
         &data_area);

MPUT NT parameter_area              /* Parameter area with the return     */
                                    /* codes and other returned data      */
MPUT NE data_area                   /* Data area with returned data or    */
                                    /* the length 0 if no data is         */
                                    /* returned                           */

PEND FI                             /* Terminate service; info is returned*/
                                    /* to the administration application  */
```

The administration application has to send a commensurate number of message parts. In the case of a UPIC client, the result may look something like this:

```
/*************************************************************************/
/* UPIC program for the administration application                       */
/*                                                                       */
/* The program sends four message parts                                  */
/*************************************************************************/

Enable_UTM_UPIC

Initialize_Conversation
[Set_TP_Name]                       /* Set TAC if necessary              */
Set_Conversation_Security_Type      /* Sign on as a UTM user             */
Set_Conversation_Security_User_ID
Set_Conversation_Security_Password


memcpy  (...)                       /* Supply all data areas             */
...
memcpy  (...)

Send_Data parameter_area            /* Send parameter area               */
Send_Data identification_area       /* Send identification area          */
Send_Data selection_ara             /* Send selection area               */
Send_Data data_area                 /* Send data area                    */

Receive parameter_area              /* Contains return codes/info        */
Receive data_area                   /* Data area containing the          */
                                    /* requested information             */

Disable_UTM_UPIC
```

For details of how to generate this kind of UPIC client, see page 128ff.

If the administration application is running on a different platform to the application being administered, the characters in the areas supplied may be converted. No problems will arise as long as these areas only contain printable characters, i.e. the identification, selection and data areas. In the parameter area (*parameter_area*), which can also contain non-printable characters and numeric values, you will need to apply a conversion mechanism.

- Define an interim parameter area in both applications which only contains printable characters.
- The administration application converts the characters in the original parameter area into printable characters, puts these in the interim parameter area and then sends this to the applications being administered.
- The administered applications write the values received to the interim parameter area, convert them to the correct numeric values and then copy these to the parameter area used for the administration call.

9 Automatic administration

You can use asynchronous programs or administration commands to administer an application automatically. This can involve having parameters raised or lowered depending on load values or triggering responses to errors. For control purposes you can, for example, use an MSGTAC program and/or time-controlled jobs.

This is how application control using the MSGTAC program proceeds:

1. An event occurs in the application and generates a message.
2. The message is passed on to the MSGTAC program.
3. MSGTAC analyses the message and then initiates the appropriate operation.

Such operations can, for instance, include calling the KDCADMI program interface, calling an administration command or starting an asynchronous administration program (FPUT/DPUT), which executes further administration tasks.

Instead of the MSGTAC program it is also possible to use a program to which a TAC is assigned that is defined as an additional message destination (KDCDEF statement MSG-DEST).

If you are using WinAdmin or WebAdmin as your administration tool, you can also use it to execute scripts or start programs when particular messages are output or when particular events occur, for instance when a threshold value is exceeded.

Another possible form of automatic administration is to have statistical data queried at regular intervals and to trigger the appropriate responses.

Diagnostic activities are yet another potential application. For certain events you can, for example, activate test mode, generate traces, create UTM dumps or have data supplied to the openSM2 event monitor.

9.1 Control using the MSGTAC program

How you can automate the administration of an application using the MSGTAC program is illustrated using an example in which the message `K041 Warning level xx for PAGEPOOL exceeded` triggers an automatic response. In place of K041 you can also insert other messages such as `K091 Due to resource bottleneck ...` for control purposes.

For this example, the message destination MSGTAC must be defined for K041, and an MSGTAC program must be written which processes this message and issues an FPUT output message to start an asynchronous program PRGK041.

You will find two versions of PRGK041 illustrated below. In one example it carries out the administration operations through the program interface and in the other it uses the command interface. The functions may also be realized within the MSGTAC routine itself.

Structure of an MSGTAC program

The MSGTAC program can be set up along the following lines:

```
/***************************** MSGTAC program ***************************/
#include <kcmsg.h>

INIT
FGET message                      /* Read message                        */
...
switch (msg-id)
   { case Kxx:...
     case K041:
       { FPUT data KCRN=PRGK041  /* Call program unit PRGK041           */
         break;
      ...
     case Kyy:..
   }
PEND FI
```

The program PRGK041 controls the operations necessitated by the occurrence of K041. The diagram below outlines what PRGK041 might look like if it uses the program interface and the command interface.

Control via the program interface

The following asynchronous administration program is started with MSGTAC.

```
/********** Program unit PRGK041 for KDCADMI program interface ***********/

#include <kcadminc.h>       /* Head file for administration             */
INIT

FGET data                    /* Read data supplied by MSGTAC             */

KDCADMI opcode=KC_GET_OBJECT
                             /* Administration call: UTM returns the     */
                             /* requested statistical data to the        */
                             /* program.                                 */

 if {...       }             /* Analyze data and prepare operations      */

KDCADMI opcode=KC_MODIFY_OBJECT
                             /* The appropriate parameter is modified.   */
                             /* Additional KDCADMI calls may be needed   */
                             /* to modify other parameters.              */

FPUT                         /* Message to administrator if necessary    */

PEND FI
```

You can have the application data read and analyzed within a program; any number of KDCADMI calls is permitted. This means that a number of application parameters can be modified if this should be necessary as a result of the current application data.

Example: activating/deactivating automatic diagnostics

The following example is a response to the message

```
K119 OSI-TP error information...
```

An MSGTAC program such as the one outlined on page 148 intercepts K119 and uses FPUT to start the administration program. Depending on the information supplied in K119, this program activates the OSI trace functions.

```
#include <kcadminc.h>       /* Header file for administration          */
...
INIT

FGET                         /* Read data from MSGTAC                   */
  if {...        }           /* Analyze data                            */

KDCADMI opcode=KC_MODIFY_OBJECT
                             /* Activate OSI trace functions under      */
                             /* certain circumstances.                  */

FPUT KCRN=admin-lterm        /* Message to administrator: trace running */

DPUT KCRN=TRACEOFF           /* After a while, a further asynchronous   */
                             /* program (TRACEOFF) deactivates the      */
                             /* trace again.                            */

PEND FI
```

You can also use this program structure, for example, to respond to the message `K065 Net message ...` You can follow the same pattern to write a program which creates a UTM dump in response to a message with KDCADMI *opcode*=KC_CREATE_DUMP.

9.2 Control via user-specific message destinations

For messages created by UTM, UTM provides four further freely available message destinations that can be used to control administrative activities. These message destinations are referred to as USER-DEST-1, USER-DEST-2,
USER-DEST-3 and USER-DEST-4 and can be explicitly assigned the following objects:

– a USER queue (the message queue of a user ID)
– a TAC queue
– an asynchronous TAC or
– an LTERM partner that is not assigned to a UPIC client

These message destinations allow you to read messages in a TAC or USER queue, for example, via the KDCS program interface using the DGET function. By means of this function and corresponding follow-up processing you can design MSGTAC-like programs that respond specifically to a message.

By assigning a USER or TAC queue to a user-specific message destination you can, for example, output UTM messages at the WinAdmin or WebAdmin administration workstation (see the openUTM manual ”Messages, Debugging and Diagnostics”).

The user-specific message destinations are configured by means of the generation statement MSG-DEST. You can obtain specific information on a message destination by means of the KC_GET_OBJECT statement and the KC_MSG_DEST_PAR object type.

You assign a message to a message destination by means of the KDCMMOD utility. The openUTM manual ”Messages, Debugging and Diagnostics” describes which messages can be assigned to the user-specific message destinations.

When a message occurs for which USER-DEST-*n* is defined as the message destination, UTM creates an asynchronous job to this message destination.

If the asynchronous job is rejected because, for example, the assigned object is disabled, the message is lost to the message destination. If there is another message for the message destination, openUTM tries again to create an asynchronous job for this message destination.

If an asynchronous TAC is assigned to a message destination USER-DEST-*n*, openUTM starts the program that is assigned to the TAC once for each message created. In contrast to the situation with MSGTAC, only one message can ever be read by means of FGET in a program run. In the KB header, KDCMSGUS is defined as the user and KDCMSGLT as the LTERM for this program unit run.

10 Access rights and data access control

Administration authorization is defined in the UTM generation. It is not bound to a certain person (user ID) or to a specific location (console). Administration can be carried out through any LTERM partner, regardless of whether this is in the form of a terminal, UPIC client or TS application. Furthermore, you can assign administration authorization to partner applications of your UTM application, allowing you to administer each your UTM applications from another application. In particular, you can administer a number of applications running on different computers centrally from one application (see the chapter “Central administration of several applications” on page 121).

In addition to general security functions (access via user IDs and the lock/key code and access list concept), openUTM also provides a special authorizations concept specially for administering a UTM application via the program interface KDCADMI and via the administration commands.

Authorization level 1

Users, clients and partner applications can call administration services which merely query, collate and analyze the information offered with regard to objects and application parameters (i.e. which only require **read** access to the configuration data) **without** any administration authorization (also referred to as administration privileges). This assumes that you have assigned the authorization level ADMIN=READ to the transaction codes via which these administration services are called.

ADMIN=READ can only be specified in the following cases:

- for the commands KDCINF, KDCINFA, KDCHELP and KDCHELPA
- for transaction codes which start program runs in which the following calls are issued:
  - KC_GET_OBJECT
  - KC_ENCRYPT with *subopcode1*=KC_READ_ACTIV_PUBLIC_KEY or *subopcode1*=KC_READ_NEW_PUBLIC_KEY
  - KC_SYSLOG with *subopcode1*=KC_INFO

In such cases, program units and transaction codes can be generated as follows:

```
B   PROGRAM ADMPROG,COMP=ILCS
B   TAC ADMTAC,PROGRAM=ADMPROG,ADMIN=READ
```

```
X/W PROGRAM ADMPROG,COMP=C
X/W TAC ADMTAC,PROGRAM=ADMPROG,ADMIN=READ
```

Authorization level 2

Administration services which modify the configuration, the application data and object properties (i.e. which require **write** access to the configuration data) can only ever be called by user IDs and partner applications **with** administration privileges (PERMIT=ADMIN). The transaction codes for these services must be configured with ADMIN=YES.

In these cases, program units and transaction codes must be generated as follows:

```
B   PROGRAM ADMPROG,COMP=ILCS
B   TAC ADMTAC,PROGRAM=ADMPROG,ADMIN=Y
```

```
X/W PROGRAM ADMPROG,COMP=C
X/W TAC ADMTAC,PROGRAM=ADMPROG,ADMIN=Y
```

The following transaction codes must be generated with ADMIN=Y:

– all administration commands, apart from KDCINF[A] and KDCHELP[A]

– transaction codes which start program runs in which KDCADMI calls other than KC_GET_OBJECT,
KC_ENCRYPT with *subopcode1*=KC_READ_ACTIV_PUBLIC_KEY or
subopcode1=KC_READ_NEW_PUBLIC_KEY or
KC_SYSLOG with *subopcode1*=KC_INFO are issued:

Other program units which call transaction codes with authorization level 2 must run under a user ID which has administration privileges.

Example

You can write an administration program which, if it is called by the transaction code ADMTAC1, merely queries whether a printer is connected to the application. If the same program is called with the transaction code ADMTAC2, the program unit again uses KC_GET_OBJECT to query whether the printer is connected to the application. However, if the printer is not connected to the application, the program unit will then also request that a connection be established to the printer (KC_MODIFY_OBJECT). ADMTAC1 can be called from any user ID and from any partner application. ADMTAC2, however, can be called only from user IDs and partner applications that have administration privileges.

The KDCDEF generation would consequently look like this:

B
```
PROGRAM ADMPROG,COMP=ILCS     (BS2000 systems)
```
X/W
```
PROGRAM ADMPROG,COMP=C        (Unix, Linux and Windows systems)
TAC ADMTAC1,PROGRAM=ADMPROG,ADMIN=READ
TAC ADMTAC2,PROGRAM=ADMPROG,ADMIN=Y
```

You can then allocate access authorizations in detail using the lock/key code and access list concept.

10.1 Configuring the administrator connection

The connection via which an administrator performs the local administration of a UTM application can be generated in different ways. It is possible to generate the connection via

- a TPOOL statement
- a PTERM and LTERM statement

Recommendation

The connection for the (main) administrator should be generated via a PTERM and an LTERM statement. On the one hand, this type of connection offers better protection against unauthorized access than an open terminal pool. On the other, an LTERM that is explicitly generated as an administrator workstation can be identified as privileged using the following statement:

```
MAX PRIVILEGED-LTERM = lterm-name
```

In bottleneck situations, UTM treats a connection generated in this way as privileged in order to make it easier for an administrator to access applications that are subject to high load.

10.2 Granting administration privileges

Administration privileges in applications with user IDs

In applications with user IDs, transaction codes for authorization level 2 can only be called under user IDs and partner applications to which administration privileges were assigned when they were entered in the configuration. User IDs and partner applications that are to administer the local application must be generated as follows:

```
USER ADMUS,[PASS=C'.....',PROTECT-PW=(...,...,...)], PERMIT=ADMIN.....

LPAP ADMPA,SESCHA=...,PERMIT=ADMIN....

OSI-LPAP ADMPAO,ASS-NAMES=...,CONTWIN=...,PERMIT=ADMIN....
```

Administration functions can also be carried out via an OSI TP partner application, if the OSI-LPAP does not have administration privileges. The application context of the OSI-LPAP must contain the abstract syntax UTMSEC in this case, and the partner has to pass on a user ID that has administration authorization in the local application.

User IDs with administration privileges can also be dynamically linked into the application configuration.

Applications without user IDs

In applications which do not have user IDs, any user or client that is connected to the application via an LTERM partner can execute administration commands and other administration TACs. Data access protection for these services can then only be implemented by means of the lock/key code and access list concept. To do so you will need to protect the administration commands with a lock code or an access list, and then only allocate a key set with a suitable key code to clients and terminals (LTERM partners) via which it should be possible to administer applications. Even in applications without user IDs, partner application can only execute administration functions with authorization level 2 if they were generated with PERMIT=ADMIN.

10.3 Generating administration commands

The openUTM administration commands you want to use when running the application must be specified during KDCDEF generation or they must be entered dynamically into the configuration using WinAdmin, WebAdmin or an administration program you have written yourself.

To do this you will need to define the administration program KDCADM with a PROGRAM statement and generate the necessary commands as KDCADM transaction codes.

An exhaustive generation of KDCADM and of all administration commands is given below. Your KDCDEF generation must only include the TAC statements for those administration commands that you want to use when running the program. The administration command KDCSHUT must be generated in all cases.

```
B   REMARK  Generate KDCADM for openUTM on BS2000 systems
B   PROGRAM KDCADM,COMP=ILCS

X/W REMARK Generate KDCADM for openUTM on Unix, Linux and Windows systems:
X/W PROGRAM KDCADM,COMP=C

    REMARK Generate dialog TACs (commands) from KDCADM:

    TAC KDCAPPL ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCBNDL ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCDIAG ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCHELP ,PROGRAM=KDCADM,ADMIN=READ        "ADMIN=Y is also permitted"
    TAC KDCINF  ,PROGRAM=KDCADM,ADMIN=READ        "ADMIN=Y is also permitted"
    TAC KDCLOG  ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCLPAP ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCLSES ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCLTAC ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCLTERM,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCPOOL ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCPROG ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCPTERM,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCSHUT ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCSLOG ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCSWTCH,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCTAC  ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCTCL  ,PROGRAM=KDCADM,ADMIN=Y
    TAC KDCUSER ,PROGRAM=KDCADM,ADMIN=Y
B   TAC KDCMUX  ,PROGRAM=KDCADM,ADMIN=Y
B   TAC KDCSEND ,PROGRAM=KDCADM,ADMIN=Y
```

```
REMARK Generate asynchronous TACs (commands) from KDCADM:

TAC KDCAPPLA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCBNDLA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCDIAGA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCHELPA,PROGRAM=KDCADM,ADMIN=READ,TYPE=A  "ADMIN=Y is also permitted"
TAC KDCINFA ,PROGRAM=KDCADM,ADMIN=READ,TYPE=A  "ADMIN=Y is also permitted"
TAC KDCLOGA ,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCLPAPA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCLSESA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCLTACA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCLTRMA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCPOOLA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCPROGA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCPTRMA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCSHUTA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCSLOGA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCSWCHA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCTACA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCTCLA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
TAC KDCUSERA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A

B TAC KDCMUXA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
B TAC KDCSENDA,PROGRAM=KDCADM,ADMIN=Y,TYPE=A
```

As with the ADMIN=READ generation above, the commands KDCINF[A] and KDCHELP[A] can be called from any user ID and from any partner application. However, you can assign a lock code to these commands (with the operand LOCK; e.g. LOCK=1). These commands can then only be called from user IDs and partner applications to which a keyset with the associated keycode (keycode 1) is assigned.

The access list concept provides another way of controlling access to these commands. An access list is assigned a key set containing a number of key/access codes, which can be for a specific group of commands, for example. If an access list like this is assigned to a command, only one user can access this command when the key set of the user's user ID and the key set of the LTERM partner via which the user is logged in each contain at least one key/access code that is also contained in the access list of the command.

You can generate the administration commands dynamically by generating the commands required using KC_CREATE_OBJECT and *obj_type* KC_TAC.

11 Program interface for administration - KDCADMI

This chapter describes the C/C++ program interface for administration. The COBOL program interface corresponds largely to the C/C++ program interface. For this reason, the following interface description will also be useful for reference if you are writing administration programs in COBOL. COBOL-specific issues that you will need to be aware of when programming in this language are described in the appendix on page 839ff.

The same C or COBOL data structures are passed to the interface in all of the supported platforms. The data fields that are irrelevant for an operating system are set to binary zero.

The C data structures are defined on Unix, Linux and Windows systems in the *kcadminc.h* header file and, on BS2000 systems, in the include element *kcadminc.h* in the library SYSLIB.UTM.064.C.

In this chapter you will find:

- a general description of a KDCADMI function call and the data areas you must pass to openUTM in the call.
- a description of the operations you can execute and the values of the parameters that need to be passed to openUTM for these operations, as well as the values returned by openUTM, for every KDCADMI operation code.

  The descriptions are ordered alphabetically according to the operation codes.
- a description of the C data structures used to pass properties of the application objects and application parameters to the program interface. This chapter begins by describing the data structures for application objects and continues with descriptions of the data structures for application parameters.

  The descriptions are arranged alphabetically by the names of the data structures.
- a detailed description of the effect of the KDCADMI call in standalone UTM applications and UTM cluster applications.

For the sake of greater clarity, this chapter uses the abbreviation UTM instead of openUTM.

11.1 Calling the KDCADMI functions

The UTM administration functions provided by the program interface for administration purposes are called using the KDCADMI function. You can pass pointers to four different data areas to UTM when calling KDCADMI. They are:

- the *parameter area* (parameter_area)

  In the parameter area you can tell UTM which operation it is to execute. This means, for example, that you can instruct UTM to return information on objects or operation parameters of the application, add an object to the configuration, change the properties of objects or delete an object.
  If the operation is to be carried out on a certain object or group of objects, then you must specify the object type of the object(s) in the parameter area.
  Once it has executed or initiated a task to carry out the operation, UTM stores the return code and the length of the data returned in the parameter area. The return code informs you whether the call was successful or unsuccessful.

- the *identification area* (identification_area)

  You require the identification area to specify the object names if, for example, an object is to be deleted from the configuration, an object's properties are to be changed or object properties are to be output. In this case, you will, in the identification area, need to pass all data required by UTM to uniquely identify the objects to be administered.

- the *selection area* (selection_area)

  In the selection area, you can pass selection criteria to UTM when querying information (see the KC_GET_OBJECT operation). UTM will then only return information on those objects meeting the selection criteria.
  Example: information on all users currently signed onto the application.

- the *data area* (data_area)

  In the data area you can pass to UTM the information that it needs, for example the names and properties of new objects if you are adding new objects to the configuration.

  UTM then returns the requested information to the program in the data area, e.g. when outputting object properties.

11.1.1 The KDCADMI function call

A C program which issues KDCADMI calls must always contain an *#include* statement referring to the header file or include element *kcadminc.h*. In *kcadminc.h*, the function KDCADMI is declared as follows:

```
void KDCADMI(struct kc_adm_parameter *  ,  /*    parameter_area       */
             void                   *  ,  /*    identification_area  */
             void                   *  ,  /*    selection_area       */
             void                   * );  /*    data_area            */
```

The KDCADMI function is called as follows:

```
#include <kcadminc.h>

KDCADMI(&parameter_area,
        &identification_area,
        &selection_area,
        &data_area );
```

where:

&parameter_area
: is the address of the parameter area named *parameter_area*.

&identification_area
: is the address of the identification area named *identification_area*.

&selection_area
: is the address of the selection area named *selection_area*.

&data_area
: is the address of the data area named *data_area*.

If one of the four areas is not needed for a particular call, then the null pointer must be passed as the address of that area.

11.1.2 Description of the data areas to be supplied

This section contains a general description of the parameters and data that can be passed to UTM when calling KDCADMI.

More detailed information concerning how to assign data to the identification area, selection area, data area and fields of the parameter area for individual operations can be found in section “KDCADMI operation codes” on page 178.

The following symbols have the following meanings:

→ The field is an input field. You can pass information to UTM using this field.

← The field is an output field. UTM returns information to the administration program in this field.

Parameter area

You can instruct UTM to perform a specific operation using the parameter area. The *opcode*, *subopcode1* and *subopcode2* fields are provided for this purpose. In the *obj_type* field, you specify the object type of the target object.

After processing, UTM stores the return code and the length of the data returned in the parameter area. You can determine if the call was successful or not from the return code.

The parameter area is defined as followed by the structure *kc_adm_parameter*:

struct kc_adm_parameter

```
int version;
KC_ADM_RETCODE retcode;
int version_data;
KC_ADM_OPCODE opcode;
KC_ADM_SUBOPCODE subopcode1;
KC_ADM_SUBOPCODE subopcode2;
KC_ADM_TYPE obj_type;
int obj_number;
int number_ret;
int id_lth;
int select_lth;
int data_lth;
int data_lth_ret;
```

Input fields in the *kc_adm_parameter* structure (hereafter indicated using the → character) that are not used must always be set to binary zero. The *version*, *version_data* and *opcode* fields must contain data every time KDCADMI is called.

The fields in the data structure have the following meanings:

→ version

Designates the version of the program interface used by the user program.

The version of the program interface indicates the variant of the program interface and the layout of the parameter areas passed at call time.

You must explicitly specify the version of the program interface on each call of KDCADMI. So far, only KC_ADMI_VERSION_1 has been defined as a version.

If the variant of the program interface is modified in a subsequent version then the version of the program interface is increased. If the extensions are compatible and you would like to continue to use the existing program interface in the new openUTM version then you do not need to adapt your existing administration programs and can continue to specify the version of the interface as KC_ADMI_VERSION_1. If you want the administration program to use the new program interface then you must adapt your programs and specify the program interface version of the current openUTM version in *version*.

The interface is designed to be source-compatible across multiple openUTM versions.

← retcode

In the *retcode* field, UTM returns the code of the function call.

There are general and function-specific return codes.
The general return codes can be returned by all functions. They are described on page 174.
The function-specific return codes only occur in connection with certain program interface calls, and they are listed in the relevant call descriptions.

If the entire length of data in the parameter area cannot be accessed, then the KDCS return code in the return area of the communication area for the service processing the KDCADMI call is assigned ’70Z’, the KCRCDC return code is assigned ’A100’, and the service is aborted with PEND ER.

The *retcode* field must be assigned the constant KC_RC_NIL before the function is called.

→ version_data

Version of the data structures used.

The version of the data structures determines the layout of the data structures used. You must specify the value of *version_data* explicitly for each KDCADMI call. The constant KC_VERSION_DATA should be used for *version_data*.

If the layout of the data structures changes in a way that is incompatible in an openUTM version, for example if the data structures receive new fields and therefore become larger, then the version number of the data structure is incremented. The constant KC_VERSION_DATA is defined in the same include file as the data structures. As a result, the correct values are entered in the *version_data* data field on recompilation.

Because the interface is source-compatible, program units must be recompiled if KC_VERSION_DATA is increased when you change over to a new UTM version. If data structures are modified to remain object-compatible then KC_VERSION_DATA is not increased and the old programs can continue to run on the new UTM version. However, it is then not possible to access new fields in data structures.

→ opcode, subopcode1, subopcode2

In these fields you tell UTM which action to execute. The *opcode* field must be assigned a value each time KDCADMI is called. This field determines which operation will be executed. In the *subopcode1* and *subopcode2* fields, you can specify in more detail what action should be taken depending on the value of *opcode*.

The values you will need to use for *opcode* to execute certain operations are summarized in the following table. The operation codes indicated by a (*) are so-called standard operations that are explained in more detail in the section "Data structures for object and parameter types" on page 170.

Function	**Value of *opcode***
B B Replace the entire application program. Replace application sections that have been marked for replacement in the Common Memory Pool. In *subopcode1* you specify whether the next highest, next lowest or the current version of the application program is to be loaded.	KC_CHANGE_APPLICATION
Create a UTM dump	KC_CREATE_DUMP
Create a new object in the configuration	KC_CREATE_OBJECT (*)
Create KDCDEF control statements online (inverse KDCDEF)	KC_CREATE_STATEMENTS
Delete an object, i.e. remove it from the configuration	KC_DELETE_OBJECT (*)
Generate, activate, delete or read RSA key pairs for data encryption ot the communication with clients	KC_ENCRYPT
Query information on objects and application parameters. You control the type and amount of detail of information returned using *subopcode1* and *subopcode2*.	KC_GET_OBJECT (*)

Function	Value of *opcode*
Permit a new sign-on for all users or for an individual user still recorded as signed on at a failed node application or who have/has a service bound to the failed node application,. Release cluster user file lock after incorrectly terminated KDCDEF run. (Only in UTM cluster applications)	KC_LOCK_MGMT
Modify object properties or application parameters	KC_MODIFY_OBJECT (*)
Import TACs, TAC queues and open asynchronous services from a terminated into a running node application (only for UTM cluster applications).	KC_ONLINE_IMPORT
Roll back transaction in PTC state.	KC_PTC_TA
B B Send a message to one dialog terminal or to all dialog terminals connected to the application.	KC_SEND_MESSAGE
Terminate an application run. Specify how the application is to be terminated (kill, normal termination) in *subopcode1* and *subopcode2* . In the case of UTM cluster applications, specify whether an individual node application or the complete UTM cluster application is to be terminated.	KC_SHUTDOWN
Establish connections to printers for which messages have been queued.	KC_SPOOLOUT
Carry out an operation on the system log file SYSLOG. You specify which operation is to be executed using *subopcode1*.	KC_SYSLOG
Update the IP address of an individual or of all communication partners. BS2000 systems: B B The communication partners must be generated with T-PROT=SOCKET. B	KC_UPDATE_IPADDR
Switch to the next generation of the user log file(s)	KC_USLOG

The information you may or must supply in the other fields of the parameter area and in the identification area, selection area and data area are dependent on the *opcode* passed. For each operation code (value of *opcode*), section “Calling the KDCADMI functions” on page 160 contains a description of the operations that can be carried out and of the information that the data area must contain to be passed to UTM in order to carry out these operations. The list is ordered alphabetically according to the operation code.

→ obj_type

The *obj_type* field must contain either the type of the target object or the type of the application parameter whose value is queried or is to be changed.

The object or parameter types that you can enter depend on which operation you require, and therefore on the values in the *opcode*, *subopcode1* and *subopcode2* fields

The two tables below contain the objects and parameter types that are supported for the standard operations in UTM. Standard operations are:

– Display
– Create
– Modify
– Delete

The column “opcode” in the table contains the operation codes for which each object type or parameter type can be specified. The following abbreviations are used:

CRE KC_CREATE_OBJECT (Create)
DEL KC_DELETE_OBJECT (Delete)
GET KC_GET_OBJECT (Show)
MOD KC_MODIFY_OBJECT (Modify)

Object types

Object type	**Value of *obj_type***	**opcode**
Abstract syntax for communication via OSI TP	KC_ABSTRACT_SYNTAX	GET
OSI TP access points for local application	KC_ACCESS_POINT	GET
Application context for communication via OSI TP	KC_APPLICATION_CONTEXT	GET
Names for the local application that were generated with KDCDEF (in a BCAMAPPL statement or in MAX APPLINAME)	KC_BCAMAPPL	GET
Names and properties of a node application in a UTM cluster application	KC_CLUSTER_NODE	GET, MOD
Connections for distributed processing via LU6.1	KC_CON	GET, CRE, DEL
Database connection	KC_DB_INFO	GET
B Edit options for screen output in line mode	KC_EDIT	GET
Global secondary storage areas for KDCS program units used to exchange data between services (GSSB)	KC_GSSB	GET

	Object type	Value of *obj_type*	opcode
	Keysets for the application. Keysets determine the access privileges of clients and users accessing services and LTERM partners.	KC_KSET	GET, MOD, CRE, DEL
	Load modules of a UTM application onBS2000 systems or the shared objects/DLLs of a UTM application on Unix, Linux or Windows systems	KC_LOAD_MODULE	GET, MOD
	LPAP partner for connecting partner applications for distributed processing via LU6.1	KC_LPAP	GET, MOD
	Sessions for distributed processing via LU6.1	KC_LSES	GET, MOD, CRE, DEL
	Local transaction codes for services provided by partner applications for distributed processing via LU6.1 or OSI TP	KC_LTAC	GET, MOD, CRE, DEL
	LTERM partner for connecting clients and printers	KC_LTERM	CRE, DEL, GET, MOD
B	User-defined message module	KC_MESSAGE_MODULE	GET
B	Multiplex connections [1]	KC_MUX	GET, MOD
	Associations with partner applications for distributed processing via OSI TP	KC_OSI_ASSOCIATION	GET
	Connections for distributed processing via OSI TP	KC_OSI_CON	GET, MOD
	OSI-LPAP partner for connecting partner applications for distributed processing via OSI TP	KC_OSI_LPAP	GET, MOD
	Transactions in PTC state	KC_PTC	GET
	Program units of the UTM application and VORGANG exits	KC_PROGRAM	CRE, DEL, GET
	Clients and printers. "Clients" can be: terminals, UPIC clients, TS applications	KC_PTERM	CRE, DEL, GET, MOD
	Temporary queues	KC_QUEUE	GET
	Allocation of UTM function keys	KC_SFUNC	GET
	Properties of sign-on procedure	KC_SIGNON	GET
X/W	IP subnets	KC_SUBNET	GET

Object type	Value of *obj_type*	opcode
Transaction codes for local services and TAC queues	KC_TAC	CRE, DEL, GET, MOD
TAC classes for the application	KC_TACCLASS	GET, MOD
LTERM pools for the application	KC_TPOOL	GET, MOD
Transfer syntax for communication via OSI TP	KC_TRANSFER_SYNTAX	GET
User IDs of the application, including queues	KC_USER	CRE, DEL, GET, MOD
User IDs of the application including their queues (optimized access for UTM cluster applications)	KC_USER_FIX, KC_USER_DYN1, KC_USER_DYN2	GET

Parameter types

Parameter type	Value of *obj_type*	opcode
Current statistics values on the capacity utilization of a UTM cluster application	KC_CLUSTER_CURR_PAR	GET, MOD
Properties of a UTM cluster application (e.g. name of the cluster filebase, node application monitoring settings) as well as current settings (e.g. number of started node applications)	KC_CLUSTER_PAR	GET, MOD
Current settings of the application parameters and statistics concerning the application capacity utilization	KC_CURR_PAR	GET, MOD
Parameters for diagnosis and UTM Accounting	KC_DIAG_AND_ACCOUNT_PAR	GET, MOD
Data for dynamic configuration: Number of existing and reserved objects, i.e. the total number of objects available in the individual object tables and the number of objects that can still be configured dynamically	KC_DYN_PAR	GET
Application name, KDCFILE name and maximum values for the application, such as the size of the cache, size and number of storage areas for KDCS program units, and the maximum number of processes permitted for the application	KC_MAX_PAR	GET, MOD
Name, type and format of a user-specific message destination	KC_MSG_DEST_PAR	GET

Parameter type	**Value of *obj_type***	**opcode**
Current page pool assignment	KC_PAGEPOOL	GET
General information on the generated temporary queues: maximum number of queues, maximum number of messages for a queue, behavior of full queues.	KC_QUEUE_PAR	GET
System parameters: Type and version of the operating system, name of the computer and the basic application data (application name, application with or without distributed processing, etc.)	KC_SYSTEM_PAR	GET
Process parameters for the application: Maximum and current number of application processes as well as of the processes available for processing asynchronous jobs and program unit runs with blocking calls.	KC_TASKS_PAR	GET, MOD
Application timer	KC_TIMER_PAR	GET, MOD
Global values for distributed processing, except for the timer defined for distributed processing	KC_UTMD_PAR	GET

Data structures for object and parameter types

For each of the object and parameter types associated with the standard operations, a data structure is provided in the header file *kcadminc.h* with which you can pass object properties and/or parameter values to UTM. There are also corresponding data structures for some of the operations that do not form part of the standard operations. UTM returns the requested information in the same data structures. The data structures are described in section “Data structures used to pass information” on page 443. The names of the data structures are created as follows:

The data structure "*typ*_str" belongs to the object or parameter type "*TYP*". For example, the data structure *kc_user_str* belongs to KC_USER, and *kc_max_par_str* to KC_MAX_PAR.

A similar principle applies to non-standard operations. E.g. the data structure *kc_application_par_str* belongs to the operation code KC_APPLICATION_PAR.

→ obj_number
: Number of objects for which the required operation is to be carried out. In *obj_number* you specify the number of objects about which UTM is to supply information when information is requested (KC_GET_OBJECT).

← number_ret
: UTM returns the actual number of objects for which the operation was carried out in *number_ret*.

→ id_lth
: In the *id_lth* field you must specify the length of the identification area *identification_area* passed in the call.
If no identification area is passed, then *id_lth*=0 must be specified.

→ select_lth
: In the *select_lth* field you must specify the length of the data structure that is passed to UTM in the selection area *selection_area*.
If no selection area is passed, then *select_lth*=0 must be specified.

→ data_lth
: In the *data_lth* field you must specify the length of the data area *data_area* passed in the call or in which UTM can return data.
If no data will be passed in the data area, then *data_lth*=0 must be specified.

← data_lth_ret
: UTM returns the actual length of the data returned in the data area in the *data_lth_ret* field.

Identification area

The identification area *identification_area* is used to identify the target object for the administration operation. All objects within a group of a certain object type must be uniquely identified by their *object_name*.

The following union is provided for passing the object name using the identification area.

union kc_id_area
`char kc_name2[2];`
`char kc_name4[4];`
`char kc_name8[8];`
`char kc_name32[32];`
`struct kc_triple_str triple;`
`struct kc_ptc_id_str ptc_id;`

Whether or not an object in the identification area needs to be uniquely specified depends on the function called.

The object name must be specified as follows in order to uniquely identify it:

- For the object types KC_CON, KC_MUX and KC_PTERM, you must specify the triplet *name, processor-name* and *bcamappl-name* as the object name, where *name* is the name of the object (for example the PTERM name), *processor-name* is the name of the computer on which the object is located, and *bcamappl-name* is the name of the local application via which the connection between the object and the application is created. These three parameters are passed to UTM using the union field *triple* of type *kc_triple_str*.

struct kc_triple_str
`char p_name[8];`
`char pronam[8];`
`char bcamappl[8];`

- For an LTERM pool (object type KC_TPOOL) you must pass the LTERM prefix, from which the names of the LTERM partners in the LTERM pool can be created, as the object name. The LTERM prefix must be passed to UTM using the *kc_name8* union field.
- For the object type KC_TACCLASS you must pass the TAC class number as the object name using the *kc_name2* union field if the function call applies to a particular TAC class. Otherwise specify binary 0 to indicate that the call applies to all TAC classes.

- For load modules, shared objects, DLLs (object type KC_LOAD_MODULE) and program units (KC_PROGRAM), pass the name specified at generation using the *kc_name32* union field.
- For the object type KC_SFUNC (UTM function keys) you must pass the short description of the function key as the object name in the union element *kc_name4*.
- For the function KC_PTC_TA (roll back a transaction in PTC state), you must fill the union element *kc_ptc_id_str* with the values from the structure *ptc_ident*. You can get the content of *ptc_ident* by first calling KC_GET_OBJECT with object type KC_PTC.

  The data structure *kc_ptc_id_str* is defined as follows:

Data structure kc_ptc_id_str

`char vg_indx[10];`
`char vg_nr[10];`
`char ta_nr_in_vg[5];`

- For the remaining object types, pass the object name specified at generation using the *kc_name8* union field if the function call applies for a particular object. Otherwise specify binary 0 to indicate that the call applies to all objects of this type.

If the identification area is not supported for a call, then you must set the area address to the null pointer. You must then set *id_lth*=0 in the parameter area.

Selection area

In the selection area you can pass a data structure containing selection criteria to UTM when querying information (operation code KC_GET_OBJECT). UTM then returns only the names and properties of the objects of the specified object type which meet the selection criteria.
The selection criteria must be passed in the data structure defined in *kcadminc.h* for that object type (*obj_type*). In the data structure you must set the search values for the fields to be used for selection.

Example

You would like to query information on which user IDs are currently signed on as users or clients. To do this, you specify the value 'Y' in the *connect_mode* field in the data structure *kc_user_str* in the selection area.

If several selection criteria are specified simultaneously, then only those objects meeting all of the selection criteria will be returned. The remaining fields in the structure must be set to binary zero. The selection criteria that can be used in a search can be found in the description of KC_GET_OBJECT starting on page 292.

If the selection area is not used for a call, then you must set the *&selection_area* area address to the null pointer. You must then set *select_lth*=0 in the parameter area.

If you want to pass selection criteria, then when calling KDCADMI, you must pass the address of the selection area and, in the *select_lth* field in the parameter area, specify the length of the data structure passed in the selection area.

Data area

The data area is used to pass object properties, parameter values and information to or from UTM. The structure of the data depends on the operation code and on the type of the target object.

If data is to be passed in the data area during a KDCADMI call, then you must pass the address of the data area and set the *data_lth* field of the parameter area to the length of the data structure passed in the data area.

If information is queried which is to be stored in the data area, then you must, when calling KDCADMI, pass the address of the data area you have provided to store the return data and set the *data_lth* field of the parameter area to the length of this data area.

If the data area is not used in a call, then you must pass the null pointer as the address of the area. You must then set *data_lth*=0 in the parameter area.

The data area must not exceed 16 MB.

11.1.3 Return codes

The KDCADMI return code consists of a main code and a subcode. The main code tells you whether the requested function has been executed or whether the execution has been initiated in a task (return code KC_MC_OK), or whether execution could not be carried out (return code not equal to KC_MC_OK). The subcode contains further information pertaining to the main code returned if the subcode is not equal to KC_SC_NO_INFO.

The code is returned in the following data structure:

```
typedef struct
{  KC_MAINCODE     mc;
   KC_SUBCODE      sc;
} KC_ADM_RETCODE;
```

UTM returns the code in the *retcode* field of the parameter area. If it is not possible to access the entire length of the parameter area or if the area is not oriented toward word boundaries, then UTM sets the return code KCRCCC ´70Z´ and the return code KCRCDC ´A100´ in the return code area of the communication area. The service is aborted with PEND ER.

Both the main codes and the subcodes are defined as enumeration type (*enum*) in the header file. By default, KDCADMI therefore returns a numeric constant.

In order to facilitate the diagnostics process when an error occurs, you can have the main codes and the subcodes issued in the form of strings (e.g. KC_MC_OK). For this, in your program, you must define the symbolic name KC_ADM_GEN_STRING using the #define statement before you include *kcadminc.h*.

```
#define KC_ADM_GEN_STRING
#include kcadminc.h
```

General return codes (independent of operation codes)

The following table lists the return codes that can be returned for any operation (i.e. for all operation codes) executed using KDCADMI. Other return codes only arise in conjunction with certain operation codes. These return codes are listed in the descriptions of the individual operation codes.

Main code = KC_MC_OK The function was executed or a task was initiated to execute the function. **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_VERS_DATA_NOT_SUPPORTED A version of the data structure which is not supported by UTM was specified in the *version_data* field of the parameter area. **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_VERSION_NOT_SUPPORTED A version of the program interface which is not supported by UTM was specified in the *version* field of the parameter area. **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_AREA_INVALID One of the data areas passed in a KDCADMI call cannot be accessed over its entire length because, for example, the area address is invalid or the required length of the area is not allocated. **Subcodes:**
KC_SC_ID_AREA The identification area cannot be accessed over its entire length.
KC_SC_SEL_AREA The selection area cannot be accessed over its entire length.
KC_SC_DATA_AREA The data area cannot be accessed over its entire length, or the address of the parameter area is within the data area.

Main code = KC_MC_NO_ADM_TAC The transaction code that initiated the administration call does not have the privileges required to execute the operation requested (administration privileges or ADM-READ privileges) **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_PAR_INVALID An invalid value was specified or a field was not set in the parameter area. **Subcodes:**
KC_SC_RETCODE The *retcode* field of the parameter area was not set to KC_RC_NIL.
KC_SC_OPCODE The operation code specified in the *opcode* field of the parameter area is invalid.
KC_SC_SUBOPCODE1 The operation modifier specified in the *subopcode1* field of the parameter area is invalid.

Main code = KC_MC_PAR_INVALID
An invalid value was specified or a field was not set in the parameter area.
Subcodes:

KC_SC_SUBOPCODE2
The operation modifier specified in the *subopcode2* field of the parameter area is invalid.

KC_SC_TYPE
The object type specified in the *obj_type* field of the parameter area is invalid.

KC_SC_NUMBER
The number of objects specified in the *obj_number* field of the parameter area is invalid.

KC_SC_ID_LTH
The length specified in the *id_lth* field of the parameter area is invalid.

Possible reasons:
– *id_lth* is not equal to the length of the name field for the object type.
– *id_lth* > 0, although no identification area may be passed.

KC_SC_SELECT_LTH
The length specified in the *select_lth* field of the parameter area is invalid.

Possible reasons:
– *select_lth* is not equal to the length of the data structure for the object type.
– *select_lth* > 0, although selection is not allowed.

KC_SC_DATA_LTH
The length specified in the *data_lth* field of the parameter area is invalid.

Possible reasons:
– *data_lth* is not equal to the length of the data structure for the object type or, for KC_GET_OBJECT, it is smaller than *obj_number* ∗ length of the data structure for the object type.
– *data_lth* > 0, but no data area was passed.
– *data_lth* > 16 MB.

KC_SC_NUMBER_RET
The *number_ret* field of the parameter area was not set to binary zero.

KC_SC_DATA_LTH_RET
The *data_lth_ret* field of the parameter area is not set to binary zero.

Main code = KC_MC_FUNCT_NOT_SUPPORTED
The operation requested is not supported by the operating system or by the version of the operating system under which the application is running.
This return code is returned by UTM when, for example, an operation has been requested in a UTM application on Unix, Linux or Windows systems that is only defined for UTM applications on BS2000 systems.
Subcode:

KC_SC_NO_INFO

11.1.4 Supplying the fields of the data structure with data when passing data

The data structure fields used in the identification area, selection area and data area to pass data between UTM and the administration programs are all of the type "char". The square brackets following the name of the field contain the length of the field. If there are no square brackets, then the field is one byte long.

The following points should be observed when passing data between UTM and an administration program:

- Names and keywords must be left-justified and any bytes left over to the right must be padded with spaces.
  The data passed to UTM can only contain uppercase letters, except for object names.

  Object names can also contain lowercase letters. The letters are not converted to uppercase. The requirements specified in section “Format and uniqueness of object names” on page 82 must be observed when creating new objects using KC_CREATE_OBJECT.

  Example: The *ptype* (*kc_pterm_str*) field is 8 bytes long. *ptype* =APPLI would be stored as follows: ´APPLI␣␣␣´.

- The numerical data returned by UTM is stored right-justified with leading spaces. Left- and right-justified numerical data is accepted when data is passed from an administration program to UTM. Right-justified entries with leading spaces or zeroes are accepted. Left-justified entries can be terminated by the null byte (\0, if the field is sufficiently large) or padded with blanks.

  Example: The *conn_users* field (*kc_max_par_str*) is 10 bytes long. *conn_users* =155 can for example be passed as follows:
  ´␣␣␣␣␣␣␣155´ or ´0000000155´ or ´155\0´ or ´155␣␣␣␣␣␣␣´

- Fields in the data structures in which no values are passed must be supplied with binary zeroes.

11.2 KDCADMI operation codes

In this section you will find an overview of the parameters you need to pass to UTM depending on the operation you wish to execute. The descriptions are organized according to the operation codes passed in the *opcode* field of the parameter area and are listed in alphabetical order.

Description format

The description of an operation code consists of four parts:

1. The first part offers a general outline of the actions that can be executed, a list of the requirements that must be fulfilled so that UTM can execute the relevant action, and notes and special cases to consider when executing the actions.

   If changes are made to the configuration and the properties then information is provided concerning the period during which the performed modifications will remain effective and whether these changes have a global or local effect for UTM cluster applications.

   If the administration function or a portion of the function described can also be executed by means of an administration command (KDCADM transaction code), then the symbol is used to indicate this command.

2. The second part is a table containing a short description of which areas (parameter, identification, selection or data area) require data for each action, and of the data that must be specified in these areas.

3. The third part consists of a schematic representation of the call, containing all optional and mandatory entries and the information that is returned by UTM. Fields requiring data before the call is made are shaded gray in the graphics. All fields in the parameter area that are not listed in the tables must be set to binary zero before you call KDCADMI.

   The symbol "—" in a table means that no data needs to be passed to UTM in this area.

4. The fourth part contains comments and notes on the graphic, i.e. regarding the entries that need to be made and the information that is returned by UTM.

11.2.1 KC_CHANGE_APPLICATION- Change application program

You can initiate the replacement of the entire application program during the application run using KC_CHANGE_APPLICATION. In this way, you can replace program units and add new program units to the application program without having to terminate the application. See the openUTM manual “Using openUTM Applications” for more information on replacing programs.

You can carry out the following operations using KC_CHANGE_APPLICATION:

B B
- Terminate a UTM application on a BS2000 system that was generated with load modules in all processes and reload it.

  B B B B
  You will need this function in standalone UTM applications to replace load modules in a common memory pool. During a reloading, the current version of the load module i loaded that has been previously specified with a KC_MODIFY_OBJECT call for the object type KC_LOAD_MODULE.

  B B
  In addition, termination of the application program in all processes and a subsequent reload will unload all load modules generated with the load mode set to ONCALL.

  B B
  Only *subopcode1*=KC_NEW and KC_SAME are possible. KC_SAME has the same effect as KC_NEW

X/W X/W X/W X/W
- An entire UTM application program on Unix, Linux or Windows systems can be replaced (*subopcode1*=KC_NEW) by the application program of the next highest file generation in the file generation directory *filebase*/PROG (*filebase*= base name of the application).

  X/W X/W X/W X/W
  You can also undo program replacement using KC_CHANGE_APPLICATION, meaning you can switch back to the previously loaded application program (*subopcode1*=KC_OLD) or you can reload the application program (*subopcode1*=KC_SAME) without switching to another file generation.

The following requirements must be met:

B B B B
- For UTM applications on a BS2000 system generated with load modules, you need to mark the parts of the application that are in a common memory pool and are to be replaced beforehand using KC_MODIFY_OBJECT calls and the KC_LOAD_MODULE object type (see page 323).

X/W X/W X/W X/W X/W
- When replacing a UTM application program on Unix, Linux or Windows systems, the different versions of the application program (including the version currently loaded) should be administered using the UTM tool KDCPROG in the file generation directory *filebase*/PROG. The file generation directory must have been created using KDCPROG (KDCPROG CREATE).

X/W
X/W
X/W

If the file generation directory *filebase*/PROG does not exist, UTM will reload the application program *filebase*/utmwork (on Unix or Linux systems) or *filebase*\utmwork (on Windows systems).

The program exchange is described in the openUTM manual “Using openUTM Applications”.

The following points should be noted when replacing the application program:

- The program units added to the new application program must have been defined at the time of the KDCDEF generation or they must have been dynamically configured by means of administration functions.
- No previously existing program units may be missing in the new application program. Jobs accepted for a transaction code for which no program unit exists after program replacement will be terminated abnormally (PEND ER) by UTM during execution.

Procedure / period of validity / transaction logging / cluster:

The call initiates program replacement, meaning that a job is created to replace the programs. The replacement itself will not have been completed when control is returned to the program unit. Program replacement is not subject to transaction logging - it cannot be undone in the same transaction by following it up with a RSET call.

Each process in the application program is replaced individually. This is done by terminating the application program running for this process and then loading the new application program. The application program is only replaced for one process at a time in order to avoid having to interrupt operations to implement program replacement. While the application program is being replaced for a given process, jobs from other processes are also being processed concurrently. These jobs may then contain processes in which the old application program is still running. This leads to a situation where jobs are processed by both the old and the new application programs during the replacement phase.

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. the application replacement is initiated in every running node application.

After the job has been processed, UTM sends you a UTM message informing you of the success or failure of the program replacement procedure. UTM sends the UTM message K074 if program replacement was carried out successfully. If UTM could not execute the program replacement, then it sends UTM message K075. If an error occurred, then UTM message K078 is sent in addition to K074 or 075. UTM message K078 contains the cause of the error as an insert.

KDCAPPL (page 666), PROG operand

Data to be supplied

	Function of the call	Data to be entered in the			
		parameter area[1]	**identification area**	**selection area**	**data area**
X/W	In UTM application on Unix, Linux and Windows systems with Shared Objects/ DLLs: Replace the current application program with the next highest version of the application program	*subopcode1*: KC_NEW	——	——	—— (A pointer to a data area to which UTM can return data must be passed in the call.)
X/W	In UTM application on Unix, Linux and Windows systems with Shared Objects/ DLLs: Undo program replacement, i.e. replace the current application program with the next lowest version of the application program	*subopcode1*: KC_OLD	——	——	
X/W	In UTM applications on Unix, Linux and Windows systems with Shared Objects/DLLs: Reload application program from the same file generation.	*subopcode1*: KC_SAME	——	——	
B	In UTM applications on BS2000 systems with load modules: Terminate the application program in all processes and then restart it in order to replace parts of the application in the common memory pool. Static application parts can therefore also be replaced when the application is linked before.	*subopcode1*: KC_NEW / KC_SAME	——	——	——

[1] The operation code KC_CHANGE_APPLICATION must be specified in the parameter area in all cases.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_CHANGE_APPLICATION
1. X/W	subopcode1	KC_NEW / KC_SAME / KC_OLD
	select_lth	0
2.	data_lth	Length of the data area / 0

Identification area

—

Selection area

—

Data area

—

KDCADMI call

KDCADMI (&parameter_area, NULL, NULL, &data_area)

Data returned by UTM

Parameter area (starting on page 184)

	Field name	Contents
3.	retcode	Return codes
4.	data_lth_ret	Actual length of the data in the data area

Data area

5. Data structure kc_change_application_str / —

1. You can use *subopcode1* to set which type of program replacement is to be executed. The following types of replacements can be carried out:

   X/W X/W X/W KC_NEW

   When replacing a UTM application on a Unix, a Linux or a Windows system, UTM loads the application program from the next highest file generation.

   B B B B B For a UTM application on a BS2000 system generated with load modules, UTM terminates the application program successively in all processes and reloads it again immediately. The current version of each of the load modules is loaded, meaning that the load modules in the common memory pool marked in KC_MODIFY_OBJECT calls are exchanged.

   B B Static application parts can therefore also be replaced when the application is linked before.

   X/W X/W KC_OLD When replacing a UTM application on Unix, Linux or Windows systems, UTM loads the application program from the next lowest file generation.

   X/W X/W In this way, the old application program can be reloaded if errors are detected in the application program after switching to a new file generation.

   KC_SAME

   X/W X/W On Unix, Linux and Windows systems, openUTM loads the application program from the same file generation.

   B On BS2000 systems, KC_SAME has the same effect as KC_NEW.

2. in the *data_lth* field you specify the length of the data area provided to contain the data returned by UTM.

   When replacing a UTM application on Unix, Linux or Windows systems, you must specify *data_lth* ≥ sizeof (*kc_change_application_str*).

   You must pass a pointer to the data area in the function call.

   B B When replacing a UTM application program under a BS000 system generated with load modules, you must set *data_lth*=0. UTM does not return any data.

3. in the *retcode* field UTM stores the return code of the call. In addition to the return codes listed in section “Return codes” on page 174, the following codes can also be returned when the application program has been replaced:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_CHANGEABLE The application was started in the dialog. Program replacement is not possible.
KC_SC_FILE_ERROR An error occurred while accessing the file generation of the application program to be loaded. UTM produced UTM message K043 with the DMS return code.
KC_SC_NOT_GEN The UTM application is gererated without load modules.
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration changes are possible since the generation of the node applications is not consistent at present.
KC_SC_JFCT_RT_CODE_NOT_OK Only for UTM cluster applications: Internal UTM error Please contact system support.

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the current time. **Subcode:**
KC_SC_CHANGE_RUNNING A program replacement is already being executed, meaning a program replacement started earlier is not yet complete.
KC_SC_INVDEF_RUNNING Only for UTM cluster applications: An inverse KDCDEF is currently running, i.e. the job cannot be processed at present.

Maincode = KC_MC_RECBUF_FULL **Subcode:**
KC_SC_NO_INFO Only for UTM cluster applications: The buffer containing the restart data is full (see openUTM manual “Generating Applications”, KDCDEF control statement MAX, RECBUF parameter).

4. In the *data_lth_ret* field of the parameter area, UTM returns the actual length of the data in the data area.

X/W 5. When replacing a UTM application on Unix, Linux or Windows systems, UTM returns the data structure *kc_change_application_str* to the data area if a pointer to a data area was passed in the KDCADMI call.

X/W
```
struct  kc_change_application_str
X/W   char program_fgg_new[4];
X/W   char program_fgg_old[4];
```

X/W program_fgg_new
X/W UTM returns the file generation number of the application program loaded
X/W as a result of program replacement.

X/W *program_fgg_old*
X/W UTM returns the file generation number of the application program loaded
X/W before program replacement was executed.

11.2.2 KC_CREATE_DUMP - Create a UTM dump

KC_CREATE_DUMP allows you to create a UTM dump for diagnostic purposes (with REASON=´DIAGDP´) without having to abort the application run.

The dump is created by the process that initiated the KDCADMI call.

Procedure / period of validity / transaction management / cluster

The call is not subject to transaction management. It has an immediate effect. The operations initiated by the call will already have been completed when control is returned to the program unit.

The following applies in UTM cluster applications:
The call applies locally to the node, i.e. a UTM dump for diagnostic purposes is only generated in this node application.

KDCDIAG (page 681), DUMP operand

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area	identification area	selection area	data area
Create a UTM dump	KC_CREATE_DUMP	——	——	——

Parameter settings

Parameter area

Field name	Contents
version	KC_ADMI_VERSION_1
retcode	KC_RC_NIL
version_data	KC_VERSION_DATA
opcode	KC_CREATE_DUMP
id_lth	0
select_lth	0
data_lth	0

Identification area

—

Selection area

—

Data area

—

KDCADMI call

KDCADMI (&parameter_area, NULL, NULL, NULL)

Data returned by UTM

Parameter area

	Field name	Contents
1.	retcode	Return codes

1. UTM only returns the codes listed in section “Return codes” on page 174.

11.2.3 KC_CREATE_OBJECT - Add objects to the configuration

KC_CREATE_OBJECT allows you to add the following objects dynamically to the application configuration:

- transport connections to remote LU6.1 applications (KC_CON)
- key sets (KC_KSET)
- LU6.1 sessions (KC_LSES)
- transaction codes by means of which service programs are started in partner applications (KC_LTAC)
- an LTERM partner to connect clients and printers (KC_LTERM)
- application program units and VORGANG exits (KC_PROGRAM)
- clients and printers (KC_PTERM)
- transaction codes and TAC queues (KC_TAC)
- user IDs, including their queues (KC_USER)

W W

openUTM on Windows systems does not support any printers.

Exactly one object can be created per KC_CREATE_OBJECT call. Within any given program unit, however, KC_CREATE_OBJECT can be called several times, i.e. several objects with the same type or with different object types can be created.

You will find more detailed information on dynamically adding objects to the configuration in chapter “Changing the configuration dynamically” on page 67.

If an object which can be dynamically generated in a **UTM cluster application** has to be deleted again then you must always delete it using the administration functions. These objects cannot be deleted simply by means of a regeneration.

Requirements for dynamically adding an object

- During KDCDEF generation of the UTM application, RESERVE was used to reserve spaces in the table for the object type; one of these spaces in the table is still empty. You can determine if there are still free spaces available in the table for the corresponding object type using KC_GET_OBJECT and the KC_DYN_PAR parameter type.
- You can only add application program units and VORGANG exits dynamically if the application was generated with load modules (BS2000 systems) or shared objects/DLLs (Unix, Linux and Windows systems). The program unit or VORGANG exit must be created by a compiler for which a program unit has already been statically configured (PROGRAM statement) during the KDCDEF generation.

B B

For ILCS-capable compilers, it is sufficient to statically generate a program unit with COMP=ILCS.

- Transaction codes for program units that use an X/Open program interface can only be added dynamically if at least one transaction code for an X/Open program unit was configured during the KDCDEF generation.
- User IDs can only be configured dynamically if the application was generated with user IDs.
  - User IDs with ID cards or can only be added dynamically if space in the table was reserved explicitly for user IDs with ID cards during the KDCDEF generation, and if one of these table spaces is still free.
  - You can only dynamically enter user IDs with Kerberos authentication if table spaces for user IDs with Kerberos authentication have been reserved explicitly and if one of these spaces is still free.

The following must be observed when adding new objects / cluster

Certain rules must be observed when adding objects that are related to each other. These rules are described in chapter “Changing the configuration dynamically” on page 67. The following are examples of objects that are related to each other:

- transaction codes and the program units and VORGANG exits assigned to them
- clients/printers and the associated LTERM partners and the connection user IDs or user IDs for the automatic KDCSIGN
- key sets referenced by user IDs, LTERM partners and transaction codes

Procedure / period of validity / transaction management / cluster

The call is subject to transaction management. Until the transaction has been completed, a dynamically created object can only be accessed within the transaction itself. Application-wide access is only possible after the transaction has been completed. In particular, the object can only be manipulated by means of administration functions after the transaction has been completed (this includes information queries). Within the same transaction, the object can only be accessed when adding additional objects that are related to it.

The call’s effects extend beyond the end of the current application run. This means that objects added dynamically are also part of the configuration for later application runs (as long as the objects are not deleted again).

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. the objects are dynamically entered in the configuration in all the node applications.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identification area**	**selection area**	**data area**
Add transport connections to the remote LU6.1 application to the configuration	*obj_type*: KC_CON	——	——	Data structure *kc_con_str* with the name and properties of the partner and the connection
Add key set to the configuration	*obj_type*: KC_KSET	——	——	Data structure *kc_kset_str* with the name and properties of the key set
Add LU6.1 session to the configuration	*obj_type*: KC_LSES	——	——	Data structure *kc_lses_str* with the name and properties of the partners involved
Add transaction code by means of which service programs are started in partner applications to the configuration	*obj_type*: KC_LTAC	——	——	Data structure *kc_ltac_str* with the name and properties of the LTAC and the partner
Add an LTERM partner to the configuration	*obj_type*: KC_LTERM	——	——	Data structure *kc_lterm_str* with the name and properties of the LTERM partner
Add a program unit or VORGANG exit to the configuration	*obj_type*: KC_PROGRAM	——	——	Data structure *kc_program_str* with the name and properties of the program unit or VORGANG exit
Add a client/printer (PTERM) to the configuration	*obj_type*: KC_PTERM	——	——	Data structure *kc_pterm_str* with the name and properties of the client/printer
Add a transaction code or TAC queue to the configuration	*obj_type*: KC_TAC	——	——	Data structure *kc_tac_str* with the name and properties of the transaction code or TAC queue
Add a user ID (including queue) to the configuration	*obj_type*: KC_USER	——	——	Data structure *kc_user_str* with the name and properties of the user ID and queue

[1] The operation code KC_CREATE_OBJECT must be specified in the parameter area in all cases.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_CREATE_OBJECT
1.	obj_type	Object type
2.	obj_number	1
	id_lth	0
	select_lth	0
3.	data_lth	Length of the data in the data area

Identification area

—

Selection area

—

Data area

4.	Data structure of the object type

KDCADMI call

```
KDCADMI (&parameter_area, NULL, NULL, &data_area)
```

Data returned by UTM

Parameter area (starting on page 230)

	Field name	Contents
5.	retcode	Return codes

1. In the *obj_type* field you must specify the type of object to be created. You can specify the following object types:

   KC_CON, KC_KSET, KC_LSES, KC_LTAC, KC_LTERM, KC_PROGRAM, KC_PTERM, KC_TAC, KC_USER.

2. Only one object can be created per call. Therefore you must set *obj_number* = 1.

3. In the *data_lth* field you specify the length of the data structure you are passing to UTM in the data area.

4. You must pass a data structure in the data area containing the name of the new object and the properties to be assigned to this object. A unique data structure is provided for each individual object type, and you must place this data structure in the data area.

   The tables on the following pages as of page 193 contain descriptions of the data structures as a function of the type of the object to be created. The table shows you which fields in the relevant data structure must be supplied with data.

   The entries in the first column of the table have the following meanings:

o	Supplying the field with data is optional
m	Supplying the field with data is mandatory
(m)	Supplying the field with data may be mandatory, depending on the data you have entered for the other mandatory parameters or at the level of the operating system under which the UTM application is running.

   Fields in the data structures that you have not explicitly specified must be set to binary zero. UTM will use the default values for these fields. You can find the default values listed in the descriptions of the data structures in section “Data structures used to pass information” on page 443.

5. In the *retcode* field UTM outputs the return codes of the call, see “Returncodes” on page 230.

11.2.3.1 obj_type=KC_CON

In order to create a new LU6.1 transport connection to a remote application, you must place the data structure *kc_con_str* in the data area.

The following table shows how the fields in the data structure are to be supplied with data.

	Field name[1]	Meaning
m	co_name[8]	Name of the partner application with which there is to be communication via the logical connection. For the format of the name see the section "Format and uniqueness of object names" on page 82. *BS2000 systems:* *co_name* can be either the BCAM name of a UTM partner application (in the case of a homogeneous link) or the name of a TRANSIT application (in the case of a heterogeneous link).
m[2] o[3]	pronam[8]	Name of the partner system. *BS2000 systems:* For *pronam* you specify the name of the processor on which the partner application *co_name* runs. This is the name of a Unix, Linux, Windows or BS2000 system.
o	bcamappl[8]	Specifies a name of the local application, as defined at generation in the control statement MAX or BCAMAPPL. A BCAMAPPL name for which T-PROT=SOCKET is generated must not be specified. Default: If nothing is specified, the primary application name in MAX ...,APPLINAME= applies.
m	lpap[8]	Name of the LPAP partner of the partner application to which the connection is to be set up. The name of the LPAP partner by means of which the partner application obtains a connection must have been defined by means of the LPAP statement at generation.
		By creating a number of CON objects with the same LPAP name, parallel connections to the partner application are configured. You must ensure that the parallel connections lead to the same partner application (*co_name* and *pronam*).
o	termn[2]	Identifier for the type of the communication partner with a maximum length of 2 characters. *termn* is not queried by UTM; it is set by the user for evaluation purposes in order, for example, to query or group terminal types. The identifier *termn* is entered in the KB header for job-receiving services (i.e. for services started in the local application by a partner application).

		Field name[1]	Meaning
X/W	o[3]	listener_port[5]	Port number of the partner application.
X/W X/W			Permissible values: 102 and 1025 - 32767 102 is only allowed if the partner application runs on a BS2000 system.
X/W X/W	o[3]	t_prot	Contains the address format with which the partner application signs on to the transport system. The address format is specified as follows:
X/W X/W			´R´ RFC1006, ISO transport protocol class 0 via TCP/IP and RFC1006 convergence protocol.
X/W	o[3]	tsel_format	Contains the format indicator of the T-selector of the partner address:
X/W			'T´ TRANSDATA format
X/W			´E´ EBCDIC character format
X/W			´A´ ASCII character format
X/W X/W			The significance of the address formats is described in the “PCMX documentation” on page 20.

[1] All fields of the *kc_con_str* data structure that are not listed and all the fields that are not relevant to the operating system used are to be set to binary zero. The data structure is described in full on page 462.

[2] Mandatory on BS2000 systems

[3] Optional on Unix, Linux and Windows systems

11.2.3.2 obj_type=KC_KSET

In order to create a new key set, you have to place the data structure *kc_kset_str* in the data area. The following table shows how the fields in the data structure are to be supplied with data.

	Field name[1]	Meaning
m	ks_name[8]	Name of the key set.
o	master	Specifies whether the key set is a master key set. A master key set contains all the key or access codes required to access the objects of the application (i.e. all key codes between 1 and the maximum value defined at KDCDEF generation in MAX KEYVALUE). 'Y' The key set is a master key set. 'N' The key set is not a master key set.
o	keys[4000]	In this field you select the key or access codes to be assigned to this key set. Only keys up to the maximum value generated (MAX KEYVALUE) can be selected. For each key to be contained in the key set, the corresponding byte in the field must be set to 1; all the *keys* fields that are not selected must contain the value 0. If the key 10 is to be created, for example, keys[9] must contain the value 1 (note: the array begins with an index of 0). A recovery buffer size of at least 16,500 bytes is recommended for 4,000 keys (MAX generation statement, RECBUF parameter).

[1] All fields of the *kc_kset_str* data structure that are not listed and all the fields that are not relevant to the operating system used are to be set to binary zero. The data structure is described in full on page 473.

11.2.3.3 obj_type=KC_LSES

In order to create a new LU6.1 session, you must place the data structure *kc_lses_str* in the data area. The following table shows how the fields in the data structure are to be supplied with data.

	Field name[1]	Meaning
m	ls_name[8]	This is the name of the session in the local application (local half-session name). The specified name must be unique and may not be assigned to any other object of name class 2. See also the section “Format and uniqueness of object names” on page 82.
m	lpap[8]	Name of the LPAP partner assigned to the partner application. *ls_name* is used for communication with the partner application assigned to the LPAP partner *lpap* in the local application.
o	rses[8]	This is the name that describes the session in the remote application (remote half-session name). The name can be up to 8 characters long.

[1] All fields of the *kc_lses_str* data structure that are not listed and all the fields that are not relevant to the operating system used are to be set to binary zero. The data structure is described in full on page 484.

11.2.3.4 obj_type=KC_LTAC

In order to create a new transaction code by means of which service programs can be started in partner applications, you must place the data structure *kc_ltac_str* in the data area. The following table shows how the fields in the data structure are to be supplied with data.

	Field name[1]	**Meaning**
m	lc_name[8]	Name of a local transaction code for the remote service program.
o	lpap[8]	Specifies the partner application to which the service program belongs. *lpap* contains the name of the LPAP partner assigned to the partner application. If *lpap* is not specified, the name of the partner application must be specified in the APRO function call (in the KCPA field).
o	rtac[64]	´The name of the associated transaction code in the remote application (*recipient_TPSU_title*).
o	rtac_lth[2]	Specifies the length of the name *rtac*. The number of relevant bytes is specified in *rtac*. Minimum value: ´1´, maximum value: ´64´
o	code_type	Specifies which code type is used by UTM internally for the *rtac* name: ´I´ INTEGER The TAC name in *rtac* is a positive integer between 0 and 67108863. *rtac* names of the code type INTEGER are only permitted for partner applications that are not UTM applications and that communicate via the OSI TP protocol. ´P´ PRINTABLE-STRING The TAC name in *rtac* is specified as a string with a maximum length of 64 characters. A distinction is drawn between uppercase and lowercase. A TAC name with the code type PRINTABLE-STRING can contain the following characters: – A, B, C, . . . , Z – a, b, c, . . . , z – 0, 1, 2, . . . , 9 – the special characters ´ - : ? = , + . ( ) / ␣ (blank) ´T´ T61-STRING *rtac* contains a T61 string. For the code type T61-STRING, UTM supports all the characters of the code type PRINTABLE-STRING as well as the following special characters: $ > < & @ # % ; ∗ _

	Field name[1]	Meaning
o	state	Specifies whether or not *lc_name* is disabled for the remote service program after the startup of the local application. 'Y' *lc_name* is not disabled. Jobs are accepted for the associated remote service. 'N' *lc_name* is disabled. Jobs are not accepted for the associated remote service.
o	accesswait_sec[5]	Maximum time waited in seconds for a session to be occupied (possibly including connection establishment) or for an association to be established after the remote service is requested (the LTAC is called). In the case of asynchronous jobs (LTAC with *ltac_type*=´A´), a wait time ≠ 0 means the job is always entered in the local message queue for the partner application. Dialog jobs are accepted. A wait time *accesswait_sec*=0 means that dialog jobs are rejected if no session/association for which the local application is the contention winner has been generated. In the case of asynchronous jobs, the FPUT call is rejected with a return code if there is no logical connection to the partner application. If there is a logical connection to the partner application, the message is entered in the local message queue. Dialog jobs are rejected regardless of the value in *accesswait_sec* if there is no logical connection to the partner application. The establishment of a connection is initiated at the same time. Minimum value: ´0´ (jobs are rejected) Maximum value: ´32767´
o	replywait_sec[5]	Maximum time in seconds waited by UTM for a reply from the remote service. By limiting the wait time you can ensure that clients or users on the terminal do not have to wait too long. *replywait_sec*=´0´ means the wait time is not limited. Minimum value: ´0´ Maximum value: ´32767´

	Field name[1]	Meaning
o	lock_code[4]	Contains the lock code assigned to the remote service in the local application (data access control). *lock_code* can contain a number between ´0´ and the maximum value defined by means of the KEYVALUE operand of the KDCDEF statement MAX. ´0´ means that the LTAC is not protected by a lock code. If *lock_code* is specified, *access_list* cannot be specified.
o	ltac_type	Specifies whether the local application processes jobs in a dialog with the remote service or whether asynchronous jobs are transferred to the partner service. ´D´ Jobs to the partner service are processed in a dialog. ´A´ The partner service is started asynchronously (by means of message queuing).
o	ltacunit[4]	Contains the number of accounting units calculated in the UTM accounting phase for each *ltac* call. The accounting units are added to the accounting unit counter of the user ID that called the *ltac*. Minimum value: ´0´, maximum value: ´4095´
o	access_list[8]	Describes a key set that specifies the access rights that a user of the local UTM application must have in order to send a job to the remote service program. Whether the job is executed in the remote application depends on the access rights defined there. The key set must be created first or already have been defined at generation. If *access_list* is specified, *lock_code* cannot be specified. A user can only access the LTAC if the key set of the user, the key set of the LTERM partner via which the user is signed on and the specified key set have at least one key code in common.

[1] All fields of the *kc_ltac_str* data structure that are not listed and all the fields that are not relevant to the operating system used are to be set to binary zero. The data structure is described in full on page 487.

11.2.3.5 obj_type=KC_LTERM

To create a new LTERM partner you must place the data structure *kc_lterm_str* in the data area. You cannot create LTERMs for bundles and groups.

The following table shows how the fields in the data structure are to be supplied with data.

	Field name[1]	Meaning
m	lt_name[8]	Name of the LTERM partner. The name may be up to 8 characters long. The name may be entered in upper or lowercase letters. The name must be unique within its name class. See section “Format and uniqueness of object names” on page 82 for information on the format and uniqueness of the name. Names of LTERM partners and transaction codes that have been deleted may not be used.
o	kset[8]	Only relevant for dialog partners (*usage_type*=´D´): Key set of the application to which the LTERM partner is to be assigned. The key set must have been created dynamically first or defined at generation. A client or client program can only start a service secured with a lock code or access list if the corresponding key or access code for the lock code or access list is contained both in the key set of the user ID under which the client or client program signs on and in the key set of the associated LTERM partner. *Note* If you do not want to define any access protection for LTERM partners in an application generated with user IDs (USER), then assign key sets to the LTERM partners containing all of the key codes of the application (MASTER).
B B B B B B B B B	o	locale_lang_id[2] locale_terr_id[2] locale_ccsname[8]	Specifies the language environment (locale) of the LTERM partner. In *locale_lang_id* you specify the language code of the language to be used when sending UTM messages to the LTERM partner. It is a maximum of 2 bytes long. In *locale_terr_id* you specify the territory code. This parameter specifies territorial particularities of the main language. It is a maximum of 2 bytes long. In *locale_ccsname* you specify the CCS name of the expanded **c**oded **c**haracter **s**et. The CCS name can be up to 8 bytes long .It must belong to one of the EBCDIC character sets defined on the BS2000 system, see XHCS User Giude.
o	lock_code[4]	Only relevant for dialog partners (*usage_type*=´D´): Lock code to be assigned to the LTERM partner (access security). The lock code must lie within the range defined in the KEYVALUE operand of the MAX KDCDEF command.

	Field name[1]	Meaning
o	state	Specifies whether the LTERM partner is to be disabled or not after generation. ´Y´ The LTERM partner is not to be disabled. (ON) ´N´ The LTERM partner is to be disabled. (OFF)
o	usage_type	Specifies whether the LTERM partner is to be configured for connecting dialog partners or for connecting printers: ´D´ LTERM partner for connecting dialog partners. ´O´ LTERM partner for connecting output media such as printers.
o	user_gen[8]	Only relevant for dialog partners (*usage_type*=´D´): For LTERM partners of terminals: User ID for which UTM will execute an automatic KDCSIGN when establishing the logical connection. This user ID must have been entered in the configuration dynamically or statically before the LTERM partner. For LTERM partners of UPIC clients and TS applications: The connection user ID must be created in the same transaction in which the LTERM partner was created. See chapter “Changing the configuration dynamically” on page 67ff for more information. Default for LTERM partners of terminals: No automatic KDCSIGN Default for LTERM partners of UPIC clients (*ptype*=´UPIC-R´) or TS applications (*ptype*=´APPLI´ or ´SOCKET´): Connection user ID with the name of the LTERM partner. If this user ID is not created explicitly in the same transaction as the LTERM partner, then UTM creates this user ID implicitly. This user ID must not already exist, however. *Note:* The use of the automatic KDCSIGN on terminals restricts access protection.
o	cterm[8]	Only relevant for LTERM partners used to connect printers (*usage_type*=´O´). Name of the printer control LTERM that is to administer the printer. The printer control LTERM must have been dynamically or statically added to the configuration before the LTERM partner (see chapter “Changing the configuration dynamically” on page 67ff). Every printer assigned to this LTERM partner (KC_PTERM) must be assigned a printer ID (*cid*) that is unique to this printer control LTERM.

		Field name[1]	Meaning
B B B B B B B B B B B B B B B	o	format_attr format_name[7]	Only relevant if a terminal is to be assigned to the LTERM partner. With the help of these fields you can assign an LTERM-specific start format to the LTERM partner. One requirement for assigning a start format is that a formatting system has been generated (KDCDEF command FORMSYS). If the start format is a #Format, then a sign-on service must also have been generated. You must always specify a *format_name* **and a** *format_attr* when defining a start format. In *format_attr* you specify the format code of the start format: ´A´ for the format attribute ATTR (+Format). ´N´ for the format attribute NOATTR (*Format). ´E´ for the format attribute EXTEND (#Format). See page 495 for descriptions of the format attributes. In *format_name* you specify the name of the start format. The name can be up to 7 characters long and may only contain alphanumeric characters.
B/X B/X B/X B/X B/X B/X B/X B/X B/X B/X B/X	o	plev[5]	Only relevant for LTERM partners of output media (*usage_type* =´O´). In *plev* you specify the control value for the message queue of the LTERM partner. As soon as the number of output jobs in the queue equals the value specified in *plev*, UTM attempts automatically to establish a connection to the printer. If a printer pool is assigned to the LTERM partner, then UTM establishes connections to all printers. UTM automatically shuts the connection down as soon as the message queue is empty. You may only specify *plev* in conjunction with *qamsg*=´Y´. *plev*=´0´ means that no control value is defined. Minimum value: ´0´ Maximum value: ´32767´
	o	qamsg	Specifies whether asynchronous jobs (FPUT and DPUT jobs) sent to the client/printer assigned to this LTERM partner are to be temporarily stored in the message queue of the LTERM partner, even if the client/printer is not connected to the application. ´Y´ An asynchronous job is added to the message queue. *qamsg*=´Y´ is not possible for *restart*=´N´. ´N´ An asynchronous job is rejected if the corresponding client/printer is not connected to the application.
	o	qlev[5]	Specifies the maximum number of asynchronous messages that may be temporarily stored in the message queue of the LTERM partner at any one time. If the control value in *qlev* is exceeded, then UTM rejects any further asynchronous jobs sent to this LTERM partner or to the client/printer assigned to it. Minimum value:´0´ Maximum value:´32767´

		Field name[1]	Meaning
	o	restart	Is only relevant for dialog partners (LTERM partners with *usage_type*=´D´). In *restart* you specify how UTM will deal with asynchronous messages in the message queue of the LTERM partner at the time when the connection is being established. ´Y´ Asynchronous messages to the client remain queued. In an application without user IDs, UTM executes an automatic service restart for this LTERM partner. In a UTM cluster application without user IDs, 'Y' is only permitted if it was generated with CLUSTER USER-RESTART=YES. ´N´ UTM deletes all asynchronous messages from the queue when the connection is established. If the job is a job complex, then a negative confirmation job is activated. UTM does not execute an automatic restart for the LTERM partner in an application without user IDs. if *qamsg*=´Y´ then *restart*=´Y´ must be set.
B B B	o	annoamsg	Only relevant for the LTERM partner of a terminal. In *annoamsg* you specify if UTM is to announce an asynchronous message to the terminal before outputting: ´Y´ Asynchronous messages are announced by a message appearing in the system line. ´N´ Asynchronous messages are output immediately (without announcement).
B B	o	netprio	Specifies the transport priority used for the transport connection between the application and the client/printer. ´M´ "Medium" transport priority ´L´ "Low" transport priority For native TCP/IP connections (*t_prot* = SOCKET) this field has no significance.

		Field name[1]	Meaning
B B B	o	kerberos_dialog	Specifies whether a Kerberos dialog is performed on the establishment of a connection for clients that support Kerberos and are connected with the application directly via via this LTERM partner (not via OMNIS). ´Y´ A Kerberos dialog is performed. ´N´ No Kerberos dialog is performed.

[1] All fields in the data structure *kc_lterm_str* that are not listed and all fields that are not relevant to the operating system you are using are to be set to binary zero. The data structure is described on page 492.

Clients/printers are assigned to LTERM partners (LTERM - PTERM) when clients/printers are being added to the configuration, or with the aid of KC_MODIFY_OBJECT.

11.2.3.6 obj_type=KC_PROGRAM

To add a new program unit or VORGANG exit to the configuration you must place the data structure *kc_program_str* in the data area.

The table below shows you how to supply data to the fields in the data structure *kc_program_str*.

		Field name[1]	Meaning
B B	m	pr_name[32]	Name of the program unit. The name may be up to 32 bytes long. You must observe the conventions in section “Format and uniqueness of object names” on page 82 when specifying a name. The name of a program unit that has been deleted from the configuration cannot be used. In UTM applications on BS2000 systems you specify the ENTRY or CSECT name of the program unit.
B B B B B B B B B B B B B B X/W X/W X/W X/W X/W X/W	(m)	compiler	Compiler or ILCS-capability of the compiler used to compile the program unit. In UTM applications on BS2000 systems the *compiler* specification is mandatory. For all program units that support ILCS you must specify ´I´ for ILCS for the compiler. In UTM applications on BS2000 systems the following settings are possible : ´I´ for ILCS (Inter Language Communication Services) ´A´ for the assembler compiler ASSEMB ´C´ for the C compiler (UTM sets this to ´I´) ´1´ for the COBOL compiler (COB1) ´F´ for the FORTRAN compiler (FOR1) ´X´ for PASCAL-XT ´P´ for PLI1 ´S´ for SPL4 In a UTM application on a Unix, Linux and Windows system the following values are possible: ´C´ for the C compiler ´+´ C++ compiler ´2´ COBOL compiler of Micro Focus ´3´ NetCOBOL compiler from Fujitsu

		Field name[1]	Meaning
B B B	m	load_module[32]	Name of the load module (BS2000 systems) or of the shared object/DLL (Unix, Linux and Windows systems) into which the program unit is linked. The name can be up to 32 characters long. The load module must be statically configured using the KDCDEF control statement LOAD-MODULE. It may not be statically linked to the application program.
X/W X/W			The shared object must/DLL be statically configured using the KDCDEF command SHARED-OBJECT.

[1] All fields in the data structure *kc_program_str* that are not listed and all fields that are not relevant to the operating system you are using are to be set to binary zero. The data structure is described in full on page 523.

11.2.3.7 obj_type=KC_PTERM

To add a printer or client (i.e. a terminal, an UPIC client or a TS application) to the configuration, you must place the data structure *kc_pterm_str* in the data area in which you will pass the name, address and properties of the client or printer to UTM. The table below shows you how to supply the fields of the structure with data.

W W

openUTM on Windows systems does not support any printers.

	Field name[1]	Meaning
m	pt_name[8]	Name of the client or printer. The name may be up to 8 characters long. The symbolic name under which the client/printer is known to the transport system should be specified in *pt_name*. See section “Format and uniqueness of object names” on page 82 for information on the format of the name and its uniqueness. Names of objects that have been deleted and which belong to the same name class may not be used. If your application contains an LTERM pool with *connect_mode*=´M´ (multi), then the triplet (*pt_name, pronam,bcamappl*) must not be the same as any naming triplet in the LTERM pool (= the triplet made up of the name of an LTERM partner in the pool, the processor name of the pool client and the BCAMAPPL name of the application which is used to establish the connection from the client). Otherwise, no other client will be able to connect via this LTERM pool. Special features of communication via the socket interface: If the connection between the communication partner and the UTM application is to be realized via the socket interface (SOCKET), and if the partner is to use a specific port number when establishing the connection, you must supply the value PRT*nnnnn* for *pt_name*, *nnnnn* being the port number in the remote system, via which the partner will establish the connection. If the partner is a UTM application, the port number cannot be supplied, because UTM does not set the port number itself. If it is only the local application that establishes the connection, and not the partner application, the name is only required internally, e.g. for administration purposes.
(m)	pronam[8]	Name of the computer on which the client/printer is located. The symbolic name under which the computer is known to the local transport system is to be specified in *pronam*. The name may be up to 8 characters long. If *ptype*=´*RSO´, then *pronam*=´*RSO´ must be specified.

B

		Field name[1]	Meaning
X/W X/W X/W		pronam[8] *(cont.)*	If the connection to the partner is established through the socket interface (TCP-IP-APPLI, *t_prot*=´T´ protocol) you must specify the system's symbolic IP address in *pronam*. On Unix, Linux and Windows systems *pronam* may be specified only with *ptype*=´UPIC-R´, ´APPLI´ or ´SOCKET´. openUTM uses the default value (blanks) for terminals and printers.
B B B	o	bcamappl[8]	Name of the UTM application through which the connection between the UTM application and the client/printer is to be established. The application name must have been statically generated using a BCAMAPPL command or during the KDCDEF generation by defining it in MAX APPLINAME. If the connection to the communication partner is to be established vie the SOCKET protocol, you must specify a BCAMAPPL name with *t_prot*=´T´. When *ptype* is not equal to ´APPLI´, ´SOCKET´ or ´UPIC-R´, only the application name generated in MAX APPLINAME (default value) may be specified for *bcamappl*.
B B W	(m)	ptype[8]	Type of client/printer You will find a list of possible types on page 531f. When *ptype*=´APPLI´, ´SOCKET´ or ´UPIC-R´, *lterm* must be specified. The specification of a *ptype* is mandatory for UTM applications on BS2000 systems. It is not permissible to specify *ptype*='PRINTER' on Windows systems.
X X X X	o	ptype_class[40]	Only relevant for printers (*ptype* = ´PRINTER´). In *ptype_class* you specify the name of the printer group (printer class) to which the printer belongs. The printer group is determined during the generation on the Unix or Linux system.
	(m)	lterm[8]	Name of the LTERM partner to be assigned to this client/printer. This parameter is optional for terminals and printers. An LTERM partner can be assigned to them at a later time using the administration functions. If the name of an LTERM partner is specified in *lterm*, then it must have been statically or dynamically added to the configuration before the terminal/printer. For UPIC clients and TS applications (*ptype* = ´UPIC-R´, ´APPLI´ or ´SOCKET´) *lterm* is a mandatory parameter. The LTERM partner specified must be created in the same transaction as the client. See "Changing the configuration dynamically" on page 67 for more information.

		Field name[1]	Meaning
	o	auto_connect	Specifies if the connection to the client/printer is to be established automatically when the application is started: ´Y´ UTM is to try to establish a connection to the client/printer every time the application is started. ´N´ UTM is not to try automatically to establish the connection. For UPIC clients, only *auto_connect*=´N´ is allowed.
	o	state	Specifies if the client/printer is to be disabled at first after being added. ´Y´ The client/printer is not be disabled (ON). ´N´ The client/printer is to be disabled (OFF).
B/X B/X B/X B/X B/X B/X	o	cid[8]	Only relevant for printers. In *cid* you specify the printer ID (CID). The CID may contain a maximum of 8 characters. The CID is required if the printer is to be administered using a printer control LTERM. The printer control LTERM identifies the printer using the CID. The CID must be unique to the printer control LTERM.
	o	map	Only relevant for TS applications (*ptype* = ´APPLI´ or ´SOCKET´): In *map* you specify if an ASCII / EBCDIC conversion is to occur when messages are exchanged. ´U´ (USER) UTM does not convert the data in the KDCS message area. ´S´, ´1´ , ´2´, ´3´, ´4´ Only allowed if communication with the TS application is realized via the socket interface. UTM converts the data in the message from EBCDIC to ASCII before transmission, and converts messages received from the partner application from ASCII to EBCDIC after they are received. It presumes that the message contains printable characters only. ´S´ and ´1´ indicate that the standard code table, converting from EBCDIC to 7-bit ASCII is to be used. ´2´, ´3´ and ´4´ stand for the three conversion tables that you can define yourself (see the openUTM manual “Generating Applications”; keyword „code conversion tables“). ´S (SYSTEM) Only permitted for TS applications (*ptype*='APPLI' or 'SOCKET'). UTM converts the data in the message from ASCII to EBCDIC before transmission, and converts messages received from the partner application from EBCDIC to ASCII.
	o	termn[2]	Code for the type of client/printer (terminal mnemonic). The code is a maximum of 2 characters long. Default values for *termn* can be found in the table on page 531 (BS2000 systems) or on page 534 (Unix, Linux and Windows systems).

		Field name[1]	Meaning
B B	o	protocol	Specifies if the NEABT user utility protocol is to be used for connections to the client/printer. ´N´ (NO) Do not use NEABT. ´S´ (STATION) Use NEABT. For clients connected through a multiplex connection, you must set *protocol* = ´S´. For UPIC clients, RSO printers and TS applications connected via the socket interface, you must set *protocol* = ´N´. In these cases, *protocol* = ´N´ is ignored.
B B	o	usage_type	Specifies whether a dialog partner or an output medium is to be configured. You can specify the following: ´D´ for a dialog partner ´O´ for an output medium (printer, for example)
B B B	o	listener_port[5]	Is only relevant when *t_prot*=´T´, ´R´ or ´L´. On BS2000 systems, *listener_port* is only allowed if communication uses the socket interface (SOCKET), in which case it is mandatory. All port numbers are permitted on BS2000 systems. You specify in *listener_port* the port number in the remote system at which the partner application awaits requests for connection establishment from outside. Possible values for *listener_port*: ´102´, ´1025´ to ´32767´
X/W X/W X/W	o	t_prot	Only relevant for clients of the type *pttype*='APPLI', 'SOCKET' or 'UPIC-R'. You specify the address format of the client's transport address. Possible values are: ´R´ RFC1006, ISO transport protocol class 0 using TCP/IP and the RFC1006 convergence protocol 'T' Native TCP-IP transport protocol for communication via the socket interface (SOCKET)
X/W X/W X/W	o	tsel_format	Only relevant for clients of the type *pttype*='APPLI', 'SOCKET' or 'UPIC-R'. You specify the format of the T-selector for the client address. Possible values are: ´T´ TRANSDATA format ´E´ EBCDIC character format ´A´ ASCII character format

	Field name[1]	Meaning
o	idletime[5]	May only be specified for dialog partners. In *idletime* you define the maximum duration in seconds which UTM waits for a response from the client after the end of a transaction or after a sign-off (KDCSIGN). If the time is exceeded, the connection to the client is closed down. If the client is a terminal, message K021 is issued before the connection is closed down. The value for idletime must not be smaller than the timer value in *kc_timer_par_str.termwait_in_ta_sec* and *kc_timer_par_str.pgwttime_sec* (see page 658). The purpose of this function is to improve data protection: If a user forgets to sign off when interrupting or finishing work at a terminal, the connection is automatically closed down when the idle time expires. This reduces the danger of unauthorized access. Maximum value: ´32767´ Minimum value: ´60´ The value 0 means wait without time limit. In the case of values smaller then 60 but not equal to 0, the value 60 is used. In the case of an invalid value, UTM sets *idletime* to the lowest value allowed and issues the return code KC_MC_OK with the subcode KC_SC_ INVALID_IDLETIME.
o	encryption_level	Only relevant for UPIC clients and also for some terminal emulations on BS2000 systems. In *encryption_level* you define the lowest encryption level for communication with a client, – whether the encryption of messages is demanded by default or not – which encryption level is demanded, – or whether the client is a “trusted” client. Possible values are: ´N´ (NONE) UTM does **not** demand data encryption by default. The client can only activate services for whose service TACs encryption was generated (see *kc_tac_str.encryption_level* on page 556), if the client agrees encryption. ´1´ (LEVEL 1) By default, UTM demands that messages are encrypted with encryption level 1. In other words, the messages are encrypted with the DES algorithm and an RSA key with a key length of 200 bits is used for exchange of the DES key. ´2´ (LEVEL 2) By default, UTM demands that messages are encrypted with encryption level 2. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 512 bits is used for exchange of the AES key.

	Field name[1]	Meaning
B B B X/W X/W X/W	encryption_level *(cont.)*	´3´ (LEVEL 3) By default, UTM demands that messages are encrypted with encryption level 3. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 1024 bits is used for exchange of the AES key. ´4´ (LEVEL 4) By default, UTM demands that messages are encrypted with encryption level 4. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 2048 bits is used for exchange of the AES key. Establishment of a connection to the client is rejected by UTM if the client does not support at least the specified encryption level (1, 2, 3, or 4). Specifying *encryption_level*=1 ... 4 is meaningful only if the encryption functionality of openUTM is installed on your system. If no encryption functionality is installed, the client cannot connect. ´T´ (TRUSTED) The client is a trusted client. Messages exchanged between the client and the application are not encrypted. A “trusted client” can activate services for which the service TACs require encryption (generated with *kc_tac_str.encryption_level* =´1´; see page 556). Select this setting only if the client is not generally accessible and communication runs through a protected connection. The following applies for the individual client types with regard to the encryption level: – Encryption levels 1 to 4 are meaningful for remote UPIC clients (PTYPE=UPIC-R). – Only encryption level 1 (*encryption_level*=1) is meaningful for clients with PTYPE= T9763 or *ANY on BS2000 systems. Levels 2, 3 and 4 are replaced by 1 by openUTM. – Encryption level 1, 2, 3 or 4 is replaced by TRUSTED by openUTM for local UPIC clients (PTYPE=UPIC-L) of an application on Unix, Linux or Windows systems. – If 1 ... 4 is specified for a partner of another type, the value is replaced by NONE by openUTM without issue of a message. For data to be encrypted on a connection to the client the corresponding RSA keys must be available. If the application is generated with OPTION GEN-RSA-KEYS=NO, KDCDEF does not create RSA keys, i.e. by default no RSA keys are available. It is however possible to transfer RSA keys by means of KDCUPD or to create them with KC_ENCRYPT. These keys can then be used by newly generated objects.

	Field name[1]	Meaning
o	usp_hdr	This parameter is only significant for PTERMs with *ptype*=´SOCKET´. It specifies the output messages for which UTM sets up a UTM socket protocol header on this connection. The possible values are: 'A' UTM creates a UTM socket protocol header for all output messages (dialog, asynchronous, K messages) and precedes the message with it. 'M' UTM creates a UTM socket protocol header for the output of K messages only and precedes the message with this. 'N' UTM does not create a UTM socket protocol header for any output message.

[1] All fields in the data structure *kc_pterm_str* that are not listed and all fields that are not relevant to the operating system you are using are to be set to binary zero. The data structure is described in full on page 529.

11.2.3.8 obj_type=KC_TAC

To create a new transaction code or a TAC queue, you must place the data structure *kc_tac_str* in the data area.

The following fields are involved in the creation of a TAC queue:

tc_name, *admin*, *qlev*, *q_mode*, *q_read_acl*, *q_write_acl*, *state* and *type*.

None of the other fields are evaluated for TAC queues.

The table below shows how to supply data to the fields in the data structure *kc_tac_str*.

	Field name[1]	**Meaning**
m	tc_name[8]	Name of the transaction code (*tac_type*='A' or 'D') or the TAC queue (*tac_type*='Q'). The name may be up to 8 characters long. See section "Format and uniqueness of object names" on page 82 for information on the format and uniqueness of the name. Names of deleted objects that belong to the same name class cannot be used.
(m)	program[32]	Name of the program unit to which the transaction code is to be assigned. The name can be up to 32 characters long. The program unit must already exist in the configuration or it must have been added before the transaction code. This parameter is not permitted for TAC queues.
o	lock_code[4]	Lock code (access security) to be assigned to the transaction code. The lock code is a whole number. It must lie within the range defined in MAX KEYVALUE during the KDCDEF generation. *Note* Jobs from a user/client will only be processed if both the key set of the user/client and the key set of the LTERM partner via which the user/client is connected to the application contain the keycode corresponding to the lock code of the service TAC.

	Field name[1]	Meaning
o	state	Specifies whether or not the transaction code or the TAC queue is to be disabled initially after generation. ´Y´ A TAC is not disabled (ON). Reading and writing are permitted for a TAC queue. ´N´ A TAC is disabled (OFF). If it is the TAC of a KDCS program unit of the type *call_type*=´B´ or ´N´, the TAC is disabled as a service TAC (1st TAC of a service) but not as a follow-up TAC of a service. Reading is permitted for a TAC queue, but not writing. ´H´ UTM does not accept any jobs for the TAC. The TAC is completely disabled (HALT). If this TAC is called as a follow-up TAC, the service is terminated with PEND ER (74Z). Asynchronous jobs that are already buffered in the message queue of the TAC are not started. They remain in the message queue until the status of the TAC is reset to ON or OFF. A TAC queue is disabled for write and read accesses. ´K´ 'K' can only be specified for asynchronous transaction codes that are also service TACs (*call_type*=´B´ or ´F´) and for TAC queues. UTM accepts jobs for the transaction code. However, the jobs are not processed; they are merely written to the job queue of the transaction code. They are processed when you change the status of the transaction code to ´Y´ or ´N´. You can use *state*=´K´ to collect jobs that are not to be executed until the application is subject to a lighter load (e.g. at night). In order to avoid overloading the page pool with too many buffered jobs, you should use the *qlev* parameter to limit the size of the job queue for the transaction code. Writing is permitted for a TAC queue, but not reading. UTM always sets *state*='Y' for the administration commands KDCSHUT and KDCTAC, even if you have entered another value. This ensures that you can administer your application at all times.

	Field name[1]	Meaning
o	tacclass[2]	Can only be specified if a TAC class was created during KDCDEF generation. In *tacclass* you specify which TAC class is to be assigned to the transaction code. You must observe the following points: – A dialog transaction code *(tac_type* = ´D´) can only be assigned a TAC class between 1 and 8 (1 ≤ *tacclass* ≤ 8). – An asynchronous transaction code (*tac_type* = ´A´) can only be assigned a TAC class between 9 and 16 (9 ≤ *tacclass* ≤ 16). – If your application is generated **without** a TAC-PRIORITIES statement, all dialog TACs (*tac_type*=´D´) from program units that use blocking calls (such as the KDCS call PGWT) must be assigned to the same dialog TAC class for which PGWT=YES must be set. Accordingly, all asynchronous TACs that use blocking calls must also be assigned to the asynchronous TAC class for which PGWT=YES is set. – If your application is generated **with** a TAC-PRIORITIES statement, all dialog TACs from program units that use blocking calls can be assigned to any dialog TAC class. You only need to set *pgwt*=´Y´. Similarly, this applies to asynchronous TACs Default (assuming that at least one TAC class exists): dialog TACs are not assigned a TAC class, asynchronous TACs are assigned TAC class 16.
o	admin	Specifies which privileges a user or client must have to be able to call this transaction code or a service containing this transaction code as the follow-up TAC. In the case of a TAC queue, the authorization refers to write and read accesses. Possible values are: ´Y´ This transaction code can only be called by a user with administration privileges. *admin*=´Y´ must be assigned to transaction codes of administration programs that do more than just read application data. In the case of a TAC queue, only a user with administration authorization can read messages from this queue or write messages to the queue. ´N´ No administration authorization is required to call the transaction code or to access the TAC queue. Program units that are started by means of a transaction code with *admin*=´N´ may not issue KDCADMI calls. ´R´ As in the case of *admin*=´N´, no administration authorization is required in order to call this transaction code or access the TAC queue. However, the associated program unit can use all the functions of KDCADMI that have read access to the application data.

	Field name[1]	Meaning
o	call_type	Specifies whether a service is started using the transaction code or if the transaction code is a follow-up TAC in a service. The following can be specified: ´B´ The TAC can be the first TAC as well as a follow-up TAC in a service (BOTH). ´F´ The TAC can only be the first TAC in a service (FIRST). ´N´ The TAC can only be a follow-up TAC in a service (NEXT).
o	exit_name[32]	Name of the VORGANG event exit to be assigned to this TAC. *exit_name* can only be specified if *call_type* = ´F´ or ´B´ has been set. The VORGANG exit specified in *exit_name* must already be contained in the configuration as a program unit of the application (dynamically with object type KC_PROGRAM or with the KDCDEF command PROGRAM). If the program unit in *program* is linked into a load module with the load mode set to ONCALL, then the VORGANG exit must be contained in the same load module.
o	qlev[5]	Only relevant for asynchronous TACs (*tac_type* = ´A´) or TAC queues (*tac_type*='Q'). UTM only takes the jobs into account at the end of the transaction. The number of messages specified in *qlev* for a message queue may therefore be exceeded when several messages are created for the same queue in a single transaction. If the number specified in *qlev* is exceeded, how UTM responds depends on the setting for *q_mode*. Minimum value: ´0´, Maximum value: ´32767´ If a value > 32767 is specified for *qlev*, then UTM will reset it to the default value without notification.
o	tac_type	Specifies whether jobs sent to this transaction code are to be processed asynchronously or in dialog mode or whether a TAC queue is created: ´D´ The transaction code is a dialog TAC ´A´ The transaction code is an asynchronous TAC 'Q' A TAC queue is created. A DPUT call can be used to write a message to a queue like this, and a DGET queue can be used to read a message from it.
o	real_time_sec[5]	Specifies the maximum amount of real time in seconds that a program unit run started with this TAC may use. If the program unit runs for a longer time, then UTM aborts the service. *real_time_sec* = ´0´ means there is no limit to the amount of real time that may be used. Minimum value: ´0´, Maximum value: ´32767´

		Field name[1]	Meaning
B B B B B B	o	cpu_time_msec[8]	Specifies the maximum amount of CPU time in milliseconds that a program unit run started with this TAC may use. If the program unit runs for a longer time, then UTM aborts the service. *cpu_time_msec* = ´0´ means there is no limit to the amount of CPU time that may be used. Minimum value: ´0´, Maximum value: ´86400000´ The values from 1 to 999 are invalid and will be rounded up to 1000 by UTM.
B B	o	dbkey[8]	Is only relevant if the program unit belonging to the transaction code sends database calls and the database system is linked to UTM. In *dbkey* you specify the database key that UTM passes to the database system when a program unit makes a database call. The format of the key depends on the database system used. The key can be up to 8 characters long. At the present time, *dbkey* is only supported for UDS. Setting *dbkey*='UTM' causes the value of the start parameter DBKEY to be passed to the database (see "Start parameters" in the openUTM manual "Using openUTM Applications").
B B B B	o	runprio[3]	Run priority of the process in the operating system in which the program unit belonging to the transaction code is running. *runprio* = ´0´ means that the transaction code is not assigned any special run priority. Minimum value: ´30´ (highest priority), Maximum value: ´255´ (lowest priority)
	o	api	UTM program interface used by the program unit belonging to the transaction code. ´K´ KDCS ´X´ X/Open interface XATMI ´C´ X/Open interface CPI-C
B B	o	satadm	Specifies if UTM SAT administration privileges are required to call the transaction code. ´Y´ The transaction code may only be called by users and partner applications that have UTM SAT administration privileges. *satadm*=´Y´ must be specified if the transaction code uses the UTM SAT administration functions. ´N´ UTM SAT administration privileges are not required to call the transaction code.

		Field name[1]	Meaning
B	o	satsel	Type of SAT logging for this transaction code. ´B´ Both successful and unsuccessful events are to be logged (BOTH). ´S´ Only successful events are to be logged (SUCC). ´F´ Only unsuccessful events are to be logged (FAIL). ´N´ No TAC-specific SAT logging is defined. Logging can only take place if SAT logging is activated for the application. (See the openUTM manual “Generating Applications” for more information on SAT logging.)
	o	tacunit[4]	Only relevant if the application uses accounting functions (see openUTM manual “Generating Applications”; Accounting and KDCDEF statement ACCOUNT and openUTM manual “Using openUTM Applications”; SAT logging). In *tacunit*, you enter the number of accounting units that will be charged to a user’s account for calling this transaction code. Only integers are allowed for *tacunit*. Minimum value: ´0´, maximum value: ´4095´
	o	pgwt	Specify only if your application processes job to TAC classes using priority control, i.e. the KDCDEF generation contains the TAC-PRIORITIES statement. In *pgwt*, you specify whether blocking calls (e.g. PGWT) can be run in a program unit started for this transaction code. ´Y´ Blocking call can be run. Specify ’Y’ only if the application is generated with TAC classes. ´N´ Blocking calls are not allowed.

	Field name[1]	Meaning
o	encryption_level	Only for service TACs (*call_type*=´F´or ´B´). In *encryption_level*, you specify whether messages for this transaction code must be encrypted or not. ´1´ (Level 1) Input messages must be encrypted using the DES algorithm for access to the transaction code. ´2´ (Level 2) Input messages must be encrypted using the AES algorithm for access to the transaction code. If *encryption_level* =´1´ or ´2´ is specified, the client can only start a service using this transaction code, if one of the following conditions is met: – The client is a “trusted“ client (see *kc_pterm_str* or *kc_tpool_str* field *encryption_level*). – The client has encrypted the input message to the transaction code with at least the specified encryption level. If a “not trusted” client does not encrypt the first input message or does not encrypt it to the required level or if the client does not support encryption, no service is started. If the transaction code is called without user data of if it is started via service concatenation, the client must be able to encrypt data, because UTM encrypts all dialog output messages it transmits and, in multi-step services, expects all input messages received from a “not trusted” client also to be encrypted. ´N´ (NONE) No message encryption required.
o	access_list[8]	You use this to specify a key set that controls the access rights of users for this transaction code. The key set must have been created dynamically beforehand or defined at generation. *access_list* must not be specified together with *lock_code*. A user can only access the transaction code if the key set of the user, the key set of the LTERM partner by means of which the user is signed on and the specified key set have at least one key code in common. If you specify neither *access_list* nor *lock_code*, the transaction code is not protected, and any user can call it.
o	q_mode	Specifies how UTM responds when the maximum number of saved but not yet executed jobs to this asynchronous TAC or to the TAC queue is reached. The possible values are: ’S’ UTM rejects any further jobs. ’W’ Only when *tac_type*=’Q’: UTM accepts further messages but deletes the oldest messages in the queue.

	Field name[1]	Meaning
o	q_read_acl[8]	Only when *tac_type*='Q': Specifies the rights (name of a key set) that a user requires in order to read and delete messages from this queue. A user only has read access to this TAC queue when the key set of the user and the key set of the logical terminal via which the user is signed on contain at least one key code that is also contained in the specified key set. If *q_read_acl* does not contain a value, all users can read and delete messages from this queue.
o	q_write_acl[8]	Only when *tac_type*='Q': Specifies the rights (name of a key set) that a user requires in order to write messages to this queue. A user only has write access to this TAC queue when the key set of the user and the key set of the logical terminal via which the user is signed on contain at least one key code that is also contained in the specified key set. If *q_write_acl* does not contain a value, all users can write messages to this queue.
o	dead_letter_q	Specifies whether a queued message should be retained in the dead letter queue if it was not processed correctly and it has not been redelivered. 'Y' Messages to this asynchronous TAC or this TAC queue which could not be processed are backed up in the dead letter queue if they are not redelivered and (with message complexes) no negative acknowledgement job has been defined. 'N' Messages to this asynchronous TAC or this TAC queue which could not be processed are not backed up in the dead letter queue. This value must be specified for all interactive TACs and for asynchronous TACs with CALL=NEXT, as well as for KDCMSGTC and KDCDLETQ.

[1] All fields in the data structure *kc_tac_str* that are not listed and all fields that are not relevant to the operating system you are using are to be set to binary zero. The data structure is described in full on page 547.

11.2.3.9 obj_type=KC_USER

To create a new user ID you must place the data structure *kc_user_str* in the data area.

A permanent queue is available to every user ID. This queue is addressed using the name of the user ID. The access of other users to this USER queue is controlled by means of the values in the *q_read_acl* and *q_write_acl* fields. The maximum number of messages that can be buffered and the response of UTM when this value is reached is determined by the values in the *qlev* and *q_mode* fields.

The table below shows you how to supply the fields of the data structure with data.

	Field name[1]	**Meaning**
m	us_name[8]	Name of the user ID. It can be up to 8 characters long. If the name of the user ID matches the name of an LTERM partner to which a UPIC client or TS application, but no user ID, has been assigned, then no user may sign on to the UTM application using this user ID. UTM then assigns this user ID exclusively to the client. See section “Format and uniqueness of object names” on page 82 for more information on the format and uniqueness of the name. Names of objects of the same name class that have been tagged for delayed delete with KC_DELAY cannot be used.
o	kset[8]	Key set of the user ID. The key set must have been created dynamically beforehand or generated statically. The key set determines the access privileges of the user/client that signs on to the application using this user ID.
o	state	Specifies if the user ID is to be disabled or not. No user/client can sign on to the application using a disabled user ID. The user ID must be released (enabled) explicitly by the administrator. ´Y´ The user ID is not to be disabled (ON). ´N´ The user ID is to be disabled (OFF).

		Field name[1]	Meaning
B B B B B B B B	o	card_position[3] card_string_lth[3] card_string_type card_string[200]	These fields are only relevant if access to the application for this user ID is only possible using a magnetic stripe card. The fields specify which subfield of the identification information on the magnetic stripe is to be checked and what information must be contained therein. Specifying *card...* excludes the possibility of specifying *certificate* and *principal*.
B			You must specify the following information in these fields:
B B B B B B			card_position Number of the byte on the magnetic stripe card where the information to be checked begins. card_string_lth Length of the identification information to be checked in bytes. Maximum value: ´100´, Minimum value: ´1´
B B B			*card_position* and *card_string_lth* must define a section of the field of identification information within the area defined by the MAX CARDLTH generation parameter.
B B			card_string_type Encoding format of the identification information to be checked:
B B B			´X´ The identification information is passed as a hexadecimal number. ´C´ The identification information is passed as a character string.
B B B B B B			card_string Character string that must be contained in the section to be checked on the magnetic stripe card. Only the length of the contents specified in *card_string_lth* is relevant if *card_string_type*= ´C´. For *card_string_type*= ´X´, the length of the relevant data is equal to 2∗*card_string_lth*.
B B			The union *kc_string* is provided for passing identification information (see page 578).

		Field name[1]	Meaning

B
B
	o	password	Password for this user ID.
			The password can be up to 8 characters long. The password specified must correspond to the complexity level specified in *protect_pw_compl* and *protect_pw_lth*. You must also specify how UTM is to interpret the data in *password* using the *password_type* field. The new password must consist of characters which are permitted in the UTM partner application, see the openUTM manual "Generating Applications", USER statement.
			Specifying *password* excludes the possibility of specifying *certificate* and *principal*.
			
			The union *kc_pw* is provided for passing the password.
			`union kc_pw`
			`char x[16]; /* for X'...' */`
			`char c[8]; /* for C'...' */`
			In UTM applications on BS2000 systems you can specify the password either as a character string or as a hexadecimal string. For a hexadecimal password (*password_type*=´X´), each half byte is displayed as a character. If you specify a password containing less than 8 characters, then you must pad *password* to the right with spaces (*password_type*= ´C´), or with the hexadecimal value for a space (*password_type*=´X´).
			In UTM applications running on Unix, Linux or Windows systems you must always pass the password as a character string (field *password.c*). If you specify a password containing less than 8 characters, then you must pad *password.c* to the right with blanks.
			You must specify *password* if *password_type* =´C´ or ´X´.
			You may not specify *password* if *password_type* = ´R´ or ´N´.
			If a user ID is to be created without a password, then you cannot specify anything in *password* and *password_type*. For *protect_pw_compl* and *protect_pw_lth*, you must set them to ´0´ (default).

B B B B B B B			
X/W X/W X/W X/W			

		Field name[1]	Meaning
B B B	o	password_type	In *password_type* you must specify how the password in *password* is to be interpreted. The following entries are possible: ´C´ The password in *password* is interpreted as a character string. ´X´ The password in *password* is interpreted as a hexadecimal password. Only allowed for user IDs in a UTM application on a BS2000 system. ´N´ No password may be specified in *password*. ´R´ The password generated is a random password. Before the user thus generated can sign on, the administrator must explicitly reset the password.
X/W X/W X/W	o	password_dark	Specifies if a password is to be hidden when entered at a terminal. ´Y´ After KDCSIGN, UTM requests the user in an interim dialog to enter the password in a darkened field. ´N´ The user conveys the password directly at KDCSIGN. The password is visible on the screen during sign-on (default value). You can also set *password_dark*=´Y´ if you have not specified a password. If the user ID is assigned a password later (with KC_MODIFY_OBJECT, for example), the password entry will be darkened. *Note* In applications running on Unix, Linux or Windows systems, password entry is never darkened.
B B B B B B B B B B B B B	o	format_attr format_name[7]	With the aid of this field you can assign the user ID a user-specific start string. You must specify *format_name* **and** *format_attr*. A requirement for assigning a start format is that a formatting system must have been generated (KDCDEF command FORMSYS). If the start format is a #Format, then a sign-on service must also have been generated. In *format_attr* you specify the format key of the start format: ´A´ for the format attribute ATTR (+Format). ´N´ for the format attribute NOATTR (*Format). ´E´ for the format attribute EXTEND (#Format). See page 495 for the meaning of the format attributes. In *format_name* you specify the name of the start format. The name can be up to 7 characters long and may only contain alphanumeric characters.

	Field name[1]	Meaning
o	locale_lang_id[2] locale_terr_id[2] locale_ccsname[8]	Language environment (locale) of the user ID. The language environment is relevant if messages and notifications from the application are to be output in different languages. See the openUTM manual “Generating Applications” for details of multi-lingual operation. In *locale_lang_id* you specify the language code of the language in which messages and notifications are to be passed. The code is a maximum of 2 bytes long. In *locale_terr_id* you specify the territorial code. It specifies territorial particularities of the language. It is a maximum of 2 bytes long. In *locale_ccsname* you specify the CCS name of the expanded character set (**c**oded **c**haracter **s**et) to be used for outputting data. The CSS name can be up to 8 characters long and must belong to a EBCDIC character set defined on the BS2000 system (see also the XHCS User Manual).
o	protect_pw_lth	Specifies the minimum number of characters a password must contain to be accepted as such by UTM (minimum length of the password). The password for a user ID can only be deleted if *protect_pw_lth* =´0´. Maximum value: ´8´, The minimum length is dependent on the complexity level specified in *protect_pw_compl*. The minimum value for *protect_pw_lth* is: ´0´ for *protect_pw_compl* = ´0´ ´1´ for *protect_pw_compl* = ´1´ ´2´ for *protect_pw_compl* = ´2´ ´3´ for *protect_pw_compl* = ´3´

	Field name[1]	Meaning
o	protect_pw_compl	Specifies the complexity level that the password for the user ID must meet. ´0´ (NONE) Any character string may be entered as the password. ´1´ (MIN) A maximum of 2 characters in a row may be identical in a password. The minimum length of a password is one character. ´2´ (MEDIUM) A maximum of 2 characters in a row may be identical in a password. The password must contain at least one letter and one number and be at least two characters long. ´3´ (MAX) A maximum of 2 characters in a row may be identical in a password. The password must contain at least one letter, one number and one special character. The minimum length is 3 characters. Special characters are all characters not between a-z, A-Z, 0-9. The space key is not a special character.
o	protect_pw_time[3]	Specifies the maximum number of days for which the password remains valid (period of validity). If *protect_pw_time* = ´0´ is specified, then the password is valid for an unlimited amount of time. Minimum value: ´0´, Maximum value: ´180´
o	restart	Specifies whether UTM saves service data for the user ID so that a service restart is possible on the next sign-on using this user ID. ´Y UTM saves service data ´N´ UTM does not save any service data.
o	permit	Specifies the administration privileges for the user ID. ´A´ (ADMIN) The user ID is to be able to execute administration functions in the local application. ´N´ (NONE) The user ID is not have any administration privileges. In UTM applications on BS2000 systems, no UTM SAT administration functions may be executed under this user ID. ´B´ (BOTH) Both administration and UTM SAT administration functions may be executed under this user ID. ´S´ (SAT) The user ID has UTM SAT administration privileges. Preselection functions may be executed.

		Field name[1]	Meaning
B	o	satsel	Specifies the type of SAT logging for the user ID.
B B B B B			´B´ Both successful and unsuccessful events are to be logged (BOTH). ´S´ Only successful events are to be logged (SUCC). ´F´ Only unsuccessful events are to be logged (FAIL). ´N´ No user-specific SAT logging is defined (NONE).
B B B B			Logging can only take place if SAT logging is activated for the application. (See the openUTM manual “Generating Applications” and openUTM manual “Using openUTM Applications” for more information on SAT logging.)
	o	protect_pw_min_time[3]	Specifies the minimum term of validity in days for the password. After changing the password, the user must not change it again before the minimum term of validity is expired. If a minimum term of 1 day is specified, the password cannot be changed again before 00.00 hrs of the following day (local time of generation). If the password is changed by the administrator or following a regeneration, the user can always change the password, regardless of whether the minimum term of validity is expired or not. *protect_pw_min_time* must not be larger than *protect_pw_time* (maximum term of validity). Minimum value: ´0´ Maximum value: ´180´
	o	qlev[5]	Specifies the maximum number of messages that can be stored temporarily in the user’s message queue. If the threshold value is exceeded, what happens depends on the value in the *q_mode* field. When *qlev*=0, no messages can be stored temporarily in the queue. When *qlev*=32767, there is no limit on the length of the queue. Minimum value: 0, maximum value: 32767
	o	q_read_acl[8]	Specifies the rights (name of a key set) that another user requires in order to be able to read and delete messages from this USER queue. Another user only has read access to this USER queue if the key set of the user’s user ID and the key set of the logical terminal via which the user is signed on each contain at least one key code that is also contained in the specified key set. If *q_read_acl* does not contain a value, all users can read and delete messages from this queue.

		Field name[1]	Meaning
	o	q_write_acl[8]	Specifies the rights (name of a key set) that another user requires in order to be able to write messages to this USER queue. Another user only has write access to this queue if the key set of the user ID and the key set of the logical terminal via which the user is signed on each contain at least one key code that is also contained in the specified key set. If *q_write_acl* does not contain a value, all users can write messages to this queue.
	o	q_mode	Specifies how UTM responds when the maximum number of not yet executed jobs in the user's queue is reached. The possible values are: 'S' UTM rejects any further jobs (default). 'W' UTM accepts further messages but deletes the oldest messages in the queue.
B B B B B B B	o	certificate[10]	Specifies whether the user has to identify himself or herself by means of a certificate when signing on to the UTM application. You will find more information in the openUTM manual "Generating Applications". Specifying *certificate* excludes the possibility of specifying *card*, *password* and *principal*. Minimum value: 0, maximum value: 2147483647
B B B B B B	o	cert_auth[10]	*cert_auth* is only permitted when *certificate* ≠ *NONE. *cert_auth* is the number of the certification point that issued the certificate. If *cert_auth* is omitted, or if 0 is specified, the number of the certification point is not checked at sign-on. Minimum value: 1, maximum value: 2147483647
B B B B B B B B	o	principal[100]	Specifies that the user is to be authenticated via Kerberos. Specifying *principal* excludes the possibility of specifying *card*, *password* and *certificate*. *principal* must be specified as an alphanumeric string in the form `windowsaccount@NT-DNS-REALM-NAME`. `windowsaccount`: Domain account of the user `NT-DNS-REALM-NAME`: DNS name of the Active Directory domain

[1] All fields in the data structure *kc_user_str* that are not listed and all fields that are not relevant to the operating system you are using are to be set to binary zero. The data structure is described in full on page 573.

11.2.3.10 Returncodes

in the *retcode* field UTM outputs the return code of the call. In addition to the return codes listed in section “Return codes” on page 174, the following codes can also be returned. Some of these return codes may arise independently of the object type specified; others only occur for certain object types.

Main code = KC_MC_DATA_INVALID A field in the data structure in the data area contains an invalid value. **Subcodes:**
KC_SC_NOT_NULL A field in the data structure that should contain a binary zero contains something else.
KC_SC_NO_INFO A field in the data structure contains an invalid value.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NAME_MISSING No name was specified for the object to be configured.
KC_SC_TAB_FULL No more objects of the specified object type can be created because the table spaces reserved during KDCDEF generation are already filled or because no table spaces for this object type have been reserved. Please note that the table spaces occupied by deleted objects are not released.
KC_SC_EXISTENT An object with this object name class already exists with the object name specified (see section “Format and uniqueness of object names” on page 82). Please note that the names of deleted objects should not be reused.
KC_SC_OBJ_DEL The object to be configured was deleted.
KC_SC_INVALID_NAME The object name begins with ’KDC’.
KC_SC_NO_CLUSTER_APPLI This action is only possible for UTM cluster applications.
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration changes are possible since the generation of the node applications is not consistent at present.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_GLOB_CRE_DEL_LOCKED Only in UTM cluster applications: It is not possible to generate an object at present because the generation or deletion of an object or the generation, deletion or activation of an RSA key pair has not yet been completed in a node application.
KC_SC_JCTL_RT_CODE_NOT_OK Only in UTM cluster applications: Internal UTM error. Please contact system support.

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the present time. **Subcode:**
KC_SC_INVDEF_RUNNING An inverse KDCDEF is currently running or an inverse KDCDEF run is being prepared (asynchronous), see KC_CREATE_STATEMENTS on page 246.

Main code = KC_MC_RECBUF_FULL The buffer containing restart information is full. The buffer size is set using the KDCDEF control statement MAX, operand RECBUF. See the openUTM manual “Generating Applications”. **Subcode:**
KC_SC_NO_INFO

Return codes for obj_type = KC_CON:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_PROCESSOR_MISSING A processor name was not specified in *pronam*. It is mandatory to specify *pronam* in UTM applications on BS2000 systems.
KC_SC_LPAP_MISSING No LPAP partner was specified.
KC_SC_LPAP_NOT_EXISTENT The specified LPAP partner does not exist.
KC_SC_BCAMAPPL_NOT_EXISTENT The application name specified in *bcamappl* does not exist.

	Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
X/W X/W	KC_SC_TPROT_NOT_ALLOWED A BCAMAPPL is referenced with *t_prot=socket*.
X/W X/W	KC_SC_INVALID_LISTENID The number specified in *listener_port* is impermissible.
X/W X/W	KC_SC_LISTENER_PORT_MISSING No *listener_port* was specified.
X/W X/W	KC_SC_INVALID_BCAMAPPL_PORT The specified port number is invalid.

Return codes for obj_type = KC_KSET:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_INVALID_KEY_VALUE An attempt was made to create more keys than are permitted by the maximum value generated in the application.

Return codes for obj_type = KC_LSES:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_LPAP_MISSING No LPAP partner was specified.

Return codes for obj_type = KC_LTAC:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_WAITTIME A negative wait time was assigned to the *waittime* parameter.
KC_SC_INVALID_LTACUNIT A value less than 0 or greater than 4095 was assigned to the *ltacunit* parameter.
KC_SC_INVALID_LOCK The *lockcode* specified in the LTAC statement is less than 0 or greater than the permitted maximum value (KDCDEF statement MAX, KEYVALUE operand).

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED *lock_code* and *access_list* cannot be specified together.
KC_SC_INVALID_ACL The specified key set does not exist.
KC_SC_INVALID_RTAC When *code*=INTEGER: The value for *recipient_TPSU_title* exceeds the max. permitted value. When *code*=PRINTABLE-STRING: The RTAC name is incorrect.
KC_SC_LPAP_NOT_EXISTENT The specified LPAP partner does not exist.
KC_SC_KSET_DEL The key referenced via *access_list* was deleted.
KC_SC_NAME_TOO_LONG The name assigned to the *rtac* parameter is too long.
KC_SC_NAME_TOO_SHORT The name assigned to the *rtac* parameter is too short.
KC_SC_INVALID_CHAR_IN_STRING The RTAC name is incorrect.

Return codes for obj_type = KC_LTERM:

Main code = KC_MC_OK The call was processed without errors. **Subcode:**
KC_SC_INVALID_LEVEL You have specified values in *plev* and/or *qlev* that exceed the maximum value allowed. The value specified is replaced by the default value.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_NAME The name specified for the object begins with "KDC". See section “Format and uniqueness of object names” on page 82 for information on object names.
KC_SC_NAME_EXISTENT The name specified for the object to be created already exists as a TAC name.
KC_SC_INVALID_FORMAT The format specified in *format_name* is a #Format, but no sign-on service was generated (there is no TAC with the name KDCSGNTC).

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NO_FORMAT_ALLOWED A start format was specified in *format_name* and *format_attr* but no formatting system was generated (KDCDEF control statement FORMSYS).
KC_SC_INVALID_FORMAT_USAGE A start format was specified in *format_name*, *format_attr* although *usage_type* =´O´ has been specified.
KC_SC_INVALID_PLEV_RESTART *plev* > ´0´ and *restart* =´N´ has been set.
KC_SC_INVALID_PLEV_QAMSG *plev* > ´0´ and *qamsg* = ´N´ has been set.
KC_SC_INVALID_PLEV_USAGE *plev* > ´0´ and *usage_type* = ´D´ has been set.
KC_SC_INVALID_RESTART_QAMSG *restart* = ´N´ and *qamsg* = ´Y´ have been set.
KC_SC_KSET_NOT_EXISTENT No key set exists for the name specified in *kset*.
KC_SC_INVALID_USAGE_CTERM The LTERM partner is to be assigned a printer control LTERM (specified in *cterm*), although *usage_type* = ´D´ has been specified (dialog partner).
KC_SC_CTERM_NOT_EXISTENT The name specified in *cterm* (printer control LTERM) does not exist.
KC_SC_CTERM_DEL The LTERM partner belonging to the name specified in *cterm* has been deleted.
KC_SC_INVALID_CTERM_USAGE The LTERM partner belonging to the name specified in *cterm* is not a dialog partner (*usage_type*=´D´).
KC_SC_INVALID_USER_USAGE The LTERM partner is to be assigned a user ID (specified in *user_gen*); however, *usage_type* is set to ´O´ (printer).
KC_SC_USER_NOT_ALLOWED A user ID is specified in the *user_gen* field, but the application was generated without user IDs.
KC_SC_KSET_DEL The referenced key set was deleted.
KC_SC_USER_NOT_EXISTENT The user ID specified in *user_gen* does not exist; the application was generated with user IDs.
KC_SC_USER_DEL The user ID specified in *user_gen* has been deleted.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_USER_NOT_ADMINISTRABLE The user ID specified in *user_gen* cannot be administered because, for example, it is a user ID that was created internally by UTM.
KC_SC_USER_ALREADY_EXISTS The application was generated without user IDs. A user ID created implicitly by UTM already exists with the name you have specified in *lt_name* (name of the LTERM partner).
KC_SC_CTERM_IS_TPOOL The object specified in *cterm* is an LTERM partner that belongs to an LTERM pool. It cannot be specified as a printer control LTERM.
B B B KC_SC_CTERM_IS_MUX The object specified in *cterm* is an LTERM partner that belongs to a multiplex connection. It cannot be specified as a printer control LTERM.
KC_SC_CTERM_IS_UTM_D The name specified in *cterm* belongs to an LPAP or OSI-LPAP partner for the purpose of connecting partner servers.
KC_SC_INVALID_LOCK The lock code specified in *lock_code* does not lie in the range between 1 and the maximum value allowed for the application (KDCDEF command MAX, KEYVALUE operand).
KC_SC_INVALID_BUNDLE_CTERM The specified CTERM is a master or slave of an LTERM bundle.
KC_SC_PRINCIPAL_AND_KERBEROS The value 'Y' in *kerberos _dialog* is not permitted if both MAX PRINCIPAL-LTH and MAX CARDLTH have the value 0.

Return codes for obj_type = KC_PROGRAM:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_LMOD_MISSING No load module / shared object / DLL was specified in *load_module*.
B B KC_SC_COMP_MISSING No compiler was specified in *compiler*.
KC_SC_LMOD_NOT_EXISTENT The load module / shared object / DLL specified in *load_module* does not exist.
KC_SC_LMOD_NOT_CHANGEABLE The load module / shared object / DLL specified in *load_module* cannot be exchanged.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_NO_LMOD The application was not generated with load modules / shared objects / DLLs. No program unit can be added dynamically to the configuration using KC_CREATE_OBJECT.
KC_SC_COMP_NOT_GEN The application does not contain a language connection module that corresponds to the compiler specified in *compiler*.
KC_SC_KDCADM_ONCALL_LMOD The default administration program KDCADM may not be created with the load mode set to ONCALL.
X/W X/W X/W KC_SC_MFCOBOL_AND_NETCOBOL It is not permitted to use programs for MFCOBOL (Micro Focus COBOL) and NETCOBOL simultaneously in a UTM application.
X/W X/W KC_SC_LANG_ENV_MISSING No language environment is available for MFCOBOL or NETCOBOL

Return codes for obj_type = KC_PTERM:

Main code = KC_MC_OK The call was processed without any errors. **Subcodes:**
KC_SC_INVALID_USAGE_APPLI_UPIC The values specified in *ptype* and *usage_type* are not compatible. *ptype* = ´UPIC-...´ was specified with *usage_type* = ´O´. The value in *usage_type* was automatically set to ´D´.
KC_SC_INVALID_IDLETIME The value of the *idletime* parameter was changed because you entered a value between 1 and 59. UTM has set *idletime* to the smallest valid value.
KC_SC_INVALID_PROTOCOL The values specified in *ptype* and *protocol* are not compatible. The following cases can arise:
– *ptype* = ´UPIC-...´ or ´*RSO´ and *protocol* = ´S´ were specified. The value in *protocol* was automatically set to ´N´. – *ptype*=´*ANY´ and *protocol* =´N´ were specified. The value in *protocol* was automatically set to ´S´.
KC_SC_INVALID_USAGE_AND_PROT The values specified in *ptype*, *protocol* and *usage_type* are not compatible. *ptype* = ´UPIC-...´ was specified with *usage_type* = ´O´ and *protocol* = ´S´. The value in *usage_type* was automatically set to ´D´, the value in *protocol* was set to ´N´.

	Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
B B B	KC_SC_PROCESSOR_MISSING No computer name was specified in *pronam*. It is mandatory to specify *pronam* in UTM applications on BS2000 systems.
B B B	KC_SC_PTYPE_MISSING No partner type was specified in *ptype*. It is mandatory to specify it for UTM applications on BS2000 systems.
	KC_SC_INVALID_NAME The object name specified begins with "KDC". This name is reserved for UTM. See section “Format and uniqueness of object names” on page 82 for information on the format of object names.
	KC_SC_INVALID_STATUS_CONNECT *state* = ´N´ was specified together with *auto_connect* = ´Y´.
	KC_SC_INVALID_PROTOCOL_USAGE *protocol* = ´N´ was specified together with *usage_type* = ´O´, and *ptype* was not assigned to ´RSO´ or ´APPLI´ or ´SOCKET´.
	KC_SC_INVALID_CID_USAGE A printer ID was specified in *cid* although *usage_type* = ´D´ (on BS2000 systems) or *ptype*=´tty´ (on Unix, Linux and Windows systems) was specified.
	KC_SC_BCAMAPPL_NOT_EXISTENT The application name specified in *bcamappl* does not exist.
X/W X/W	KC_SC_INVALID_BCAMAPPL_PORT Invalid listener port
	KC_SC_INVALID_BCAMAPPL_PTYPE The name specified in *bcamappl* is not identical to the application name (APPLINAME) defined in the KDCDEF control statement MAX, although *ptype* ≠ ´APPLI´, ´SOCKET´ or ´UPIC-R´.
	KC_SC_LTERM_NOT_EXISTENT The LTERM partner specified in *lterm* does not exist.
	KC_SC_PTYPE_NO_LTERM *ptype* = ´APPLI´, ´SOCKET´ or ´UPIC-...´ was specified, but no LTERM partner was specified in *lterm*.
	KC_SC_INVALID_USAGE_LTERM The value specified in *usage_type* is not compatible with the LTERM partner specified in *lterm*.
	KC_SC_INVALID_BUNDLE_USAGE *usage_type*=´O´ not permitted for bundle
	KC_SC_INVALID_BUNDLE *usage_type*=´D´ was specified and an LTERM partner was specified in *lterm* that already has been assigned a client.

	Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
	KC_SC_INVALID_GROUP_USAGE *usage_type*=´O´ not permitted for group
	KC_SC_INVALID_PROV_BUNDLE *usage_type*=´D´ was specified and an LTERM partner was specified in *lterm* that already has been assigned a client in this transaction.
	KC_SC_LTERM_DEL The LTERM partner specified in *lterm* has been deleted.
	KC_SC_CID_MISSING No data was specified in *cid*: The LTERM partner specified in *lterm* is assigned a printer control LTERM (specified in *cterm*). A printer ID must then be specified for the printer.
	KC_SC_INVALID_CID The printer ID specified in *cid* already belongs to another printer that has been assigned to the same printer control LTERM.
	KC_SC_CTERM_DEL The printer control LTERM of the LTERM partner specified in *lterm* has been deleted.
	KC_SC_USRT_TAB_FULL For *ptype* = ´APPLI´, ´SOCKET´ or ´UPIC-...´: UTM cannot create a connection user ID because all table spaces reserved for user IDs during generation have been used.
X/W X/W X/W	KC_SC_PROCESSOR_NOT_ALLOWED The name of a computer was specified in *pronam* although *ptype* = ´TTY´, ´PRINTER´ or ´UPIC-L´ was specified.
	KC_SC_INVALID_MAP_PTYPE *map*= ´S´ was specified although *ptype* ¼ ´APPLI´ or ´SOCKET´ was specified.
X/W X/W	KC_SC_INVALID_CONNECT_PTYPE *auto_connect*=´Y´ was specified together with *ptype* = ´TTY´ or ´UPIC-...´.
	KC_SC_INVALID_AUTOUSER_PTYPE *ptype* = ´APPLI´, ´SOCKET´ or ´UPIC-...´: The connection user ID (*user_gen*) defined for the LTERM partner specified in *lterm* is not created in the same transaction.
	KC_SC_INVALID_LTERM_PTYPE *ptype*= ´APPLI´, ´SOCKET´ or ´UPIC-...´: The LTERM partner specified in *lterm* is not created in the same transaction.
	KC_SC_LTERM_IS_TPOOL The LTERM partner specified in *lterm* belongs to an LTERM pool.
B B B	KC_SC_LTERM_IS_MUX The LTERM partner specified in *lterm* belongs to a multiplex connection, i.e. it has been created implicitly by UTM for a multiplex connection.

	Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
	KC_SC_LTERM_IS_UTM_D The name specified in *lterm* belongs to an LPAP or OSI-LPAP partner for connecting partner servers.
	KC_SC_LTERM_IS_MASTER The specified LTERM is a master Lterm.
	KC_SC_LTERM_IS_ALIAS The specified LTERM is an alias Lterm.
	KC_SC_INVALID_GROUP_PTYPE The specified LTERM is a primary Lterm and the PTYPE is not APPLI or SOCKET.
	KC_SC_INVALID_LTERM_SLAVE_PTYP The specified LTERM is a slave Lterm and the PTYPE is not APPLI or SOCKET.
	KC_SC_INVALID_LTERM_PTYPE Different PTYPEs within a bundle.
	KC_SC_INVALID_APPLI_USER *ptype* = ´APPLI´, ´SOCKET´ or ´UPIC-R´: For the LTERM partner specified in the *lterm* field, no connection user ID has been specified, i.e. *user_gen* was not specified when the LTERM partner was added. A user ID with the name of the LTERM partner exists, but it was not created in the same transaction as the client (see page 74f).
	KC_SC_INVALID_LISTENID The number specified in *listener_port* is invalid.
B B	KC_SC_PRONAM_NOT_RSO ´∗RSO´ was specified in *ptype*, but *pronam* was not set to ´*RSO´.
B B	KC_SC_PTYPE_NOT_RSO ´∗RSO´ was specified in *pronam*, but *ptype* was not set to ´*RSO´.
	KC_SC_INVALID_USAGE_APPLI_UPIC *ptype*=´APPLI´, ´SOCKET´ or ´UPIC-...´ was specified with USAGE=´O´.
	KC_SC_INVALID_IDLETIME_USAGE *idletime* was specified for an output station.
	KC_SC_INVALID_AUTOUSER_PTYPE *ptype* =´APPLI´, ´SOCKET´ or ´UPIC-...´ was specified, but the USER with the name of the specified LTERM is not created by the same transaction.
W W W	KC_SC_PRINTER_NT_NOT_SUPPORTED *ptype*=´PRINTER´ was specified in the UTM application on Windows systems, however, openUTM on Windows systems does not support printers.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_PTYPE_AND_PROT The PTERM has not been generated with *ptype*=SOCKET and the referenced BCAMAPPL has been generated with TCP/IP. B B BS2000 systems: The PTERM has been generated with *ptype*=SOCKET and the referenced BCAMAPPL has not been generated with TCP/IP.
X/W X/W X/W KC_SC_INVALID_TPROT_AND_TPROT The PTERM referenced with *ptype*=SOCKET and the referenced BCAMAPPL is not generated with TCP/IP.
KC_SC_INVALID_USP_AND_PROT A value not equal to NO is contained in the *usp_hdr* field, and the referenced BCAMAPPL does not have TCP/IP.
KC_SC_TPROT_NOT_ALLOWED Transport protocol not permitted.
KC_SC_KEY_NOT_GEN_CREA_IT An encryption level for which no RSA key pair was created at generation was selected in the *encryption_level* field. If a PTERM is to be created with this encryption level, you must first dynamically generaste an RSA key pair with the desired encryption level. Note that this can take quite a long time for encryption levels 3 and 4.

Return codes for obj_type = KC_TAC:

Main code = KC_MC_OK The call was processed without error. **Subcode:**
KC_SC_INVALID_VALUE One or more of the following values were invalid or were set automatically: – A number was specified in *qlev* that is larger than the maximum number permitted. UTM replaced the value with the maximum value. – A time between ´1´ and ´999´ msec was specified in *cpu_time_msec*. The time was set to ´1000´. T – A time was specified in *cpu_time_msec* that is larger than the maximum value permitted. The value was replaced with the maximum value. – A time was specified in *real_time_sec* that is larger than the maximum value permitted. The value was replaced with the maximum value. – A priority between ´1´ and ´29´ was specified in *runprio*. The value was set to ´30´. – A value was specified in *tacunit* that is larger than the maximum value allowed. The value was replaced with the maximum value.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED The specification of *lock_code* and *access_list* together is not permitted.
KC_SC_PROGRAM_MISSING No data was entered in *program*.
KC_SC_INVALID_TYPE No queues are permitted in UTM-FF.
KC_SC_INVALID_NAME You tried to generate an administration TAC without setting admin='Y' or the TAC name (*tc_name*) begins with "KDC". These names are reserved for UTM. See section “Format and uniqueness of object names” on page 82 for information on the format of object names.
KC_SC_TACUNIT_ILL Invalid value for *tacunit*.
KC_SC_PROGRAM_NOT_EXISTENT The program unit specified in *program* does not exist.
KC_SC_INVALID_EXIT_PROGRAM The VORGANG exit specified in *exit_name* belongs to a load module / shared object / DLL generated with the load mode set to ONCALL. However, this load module does not contain the program unit specified in *program*.
KC_SC_NAME_EXISTENT The transaction code specified in *tc_name* is already defined as an LTERM partner. The names of transaction codes and LTERM partners belong to the same name class (see section “Format and uniqueness of object names” on page 82).
KC_SC_EXIT_NEXT_TAC A VORGANG exit was specified in *exit_name* although the transaction code should have been configured as a follow-up (next) TAC (*call_type*=´N´).
KC_SC_PROGRAM_DEL The program unit specified in *program* has been deleted.
KC_SC_EXIT_NOT_EXISTENT The VORGANG exit specified in *exit_name* does not exist.
KC_SC_INVALID_TCBENTRY Specifying *tcbentry* is not allowed.
KC_SC_EXIT_DELETED The VORGANG exit specified in *exit_name* has been deleted.
KC_SC_XOPEN_NOT_ALLOWED A value not equal to ´K´ (KDCS) was specified in *api* and the application was generated without X/Open TACs. You can only dynamically configure a transaction code for a program unit that uses the X/Open program interface functions if at least one transaction code of this type was statically generated with KDCDEF.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_QMODE *q_mode*=´W´ is only permitted for TAC queues.
KC_SC_INVALID_QMODE_QLEV *q_mode*=´W´ but *qlev* is not between 1 and 32766.
KC_SC_INVALID_QMODE_FF Invalid *q_mode* for UTM-FF.
KC_SC_KSET_DEL The key set referenced via *kset* or *access_list* was deleted.
KC_SC_READ_ACL_DEL The key set referenced via *q_read_acl* was deleted.
KC_SC_WRITE_ACL_DEL The key set referenced via *q_write_acl* was deleted.
KC_SC_INVALID_LOCK The lock code specified in *lock_code* is not between 1 and the maximum value (KEYVALUE operand of the MAX command) allowed for the application.
KC_SC_INVALID_TACCLASS The data specified in *tacclass* and *tac_type* is incompatible:
– *tac_type*=´D´ (dialog TAC) was specified and a value was specified in *tacclass* that is not between ´1´ and ´8´. – *tac_type*=´A´ (asynchronous TAC) was specified and a value was specified in *tacclass* that is not between ´9´ and ´16´.
KC_SC_NO_TACCLASS_GENERATED Data was specified in the *tacclass* field, but the application was generated without TAC classes.
KC_SC_PGWT_TACCLASS 'Y' was specified in *pgwt*. That is not allowed if the TAC-PRIORITIES statement was issued, during the KDCDEF generation.
KC_SC_PGWT_NO_PGWT_TASKS 'Y' was specified in *pgwt*, but MAX TASKS-IN-PGWT=0 (default) was specified in KDCDEF generation of the application.
KC_SC_ILLEGAL_STATUS 'K' (Keep) was specified in *state*, although *tac_type*=´D´ (i.e. the transaction code is not an asynchronous TAC) and/or *call_type*≠´F´or ´B´ (the transaction code is not defined as the first TAC of a service).
KC_SC_PGWT_YES_NO_TACCLASS You entered 'Y' for *pgwt*, although the application was generated without TAC classes.
KC_SC_CALLTYPE_N_ENCRYPT You entered '1' for *encryption_level* ´1´, although the TAC is not a service TAC, i.e. *call_type*=´N´.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_READ_ACL The key set specified in *q_read_acl* does not exist.
KC_SC_INVALID_WRITE_ACL The specified key specified in *q_write_acl* set does not exist.
KC_SC_INVALID_ACL The specified key set specified in *access_list* does not exist.
KC_SC_DLETQ_YES_NOT_ALLOWED Invalid value for *dead_letter_q*.

Return codes for obj_type = KC_USER:

Main code = KC_MC_OK The call was processed without error. **Subcode:**
KC_SC_INVALID_PROTECT_PW The value specified in *protect_pw_lth* and/or in *protect_pw_time* were larger than the maximum value allowed. The value was set to the maximum value.

	Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
	KC_SC_CARD_TAB_FULL The table space reserved for CARD during KDCDEF generation is already occupied or no table spaces were reserved for CARD.
X/W X/W	KC_SC_NO_CARD_ALLOWED *card*... was specified even though no formatting has been generated.
	KC_SC_INVALID_NAME The user ID (*us_name*) specified begins with "KDC". These names are reserved for UTM. See section “Format and uniqueness of object names” on page 82 for information on the format of the object names.
B B B	KC_SC_INVALID_FORMAT The start format specified in *format_name* and *format_attr* is a #Format, but no sign-on service was generated (there exists no TAC with the name KDCSGNTC).
B B B	KC_SC_NO_FORMAT_ALLOWED A start format was specified in *format_name* and *format_attr*, but no formatting system was generated (KDCDEF control statement FORMSYS).

	Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
	KC_SC_COMPL_MISSING The password specified in *password* does not meet the complexity level required in *protect_pw_compl*.
	KC_SC_KSET_NOT_EXISTENT No key set exists for the name specified in *kset*.
B B	KC_SC_INVALID_POSITION The value specified in *card_position* is invalid.
	KC_SC_MIN_LTH_WITHOUT_PASSWORD No password was specified in *password* although *protect_pw_lth* > ´0´ is set.
	KC_SC_APPLICATION_WITHOUT_USER You cannot create a user ID because the application was generated without user IDs.
	KC_SC_INVALID_READ_ACL The key set specified in *q_read_acl* does not exist.
	KC_SC_INVALID_WRITE_ACL The specified key specified in *q_write_acl* set does not exist.
	KC_SC_INVALID_QMODE_QLEV *q_mode*=´W´ but *qlev* is not between 1 and 32766
	KC_SC_INVALID_QMODE_FF Invalid *q_mode* for UTM-FF
	KC_SC_INVALID_CERTIFIC_AUTH A certificate number was specified for a USER without a certificate.
	KC_SC_INVALID_CERTIFICATE It is not possible to create a USER with both a certificate and a password.
	KC_SC_KSET_DEL The key set referenced via *kset* was deleted.
	KC_SC_READ_ACL_DEL The key set referenced via *q_read_acl* was deleted.
	KC_SC_WRITE_ACL_DEL The key set referenced via *q_write_acl* was deleted.
B B B	KC_SC_INVALID_PRINCIPAL A principal was specified and at the same time the CARD, PASSWORD or CERTIFICATE parameter was specified.
	KC_SC_INVALID_QLEV_FF Invalid *qlev* for UTM-FF
B B	KC_SC_PRINCIPAL_AND_PW It is not possible to generate a USER with both a principal and a password.

	Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
B B	KC_SC_PRINCIPAL_AND_CARD It is not possible to generate a USER with both a principal and a chip card.
B B B	KC_SC_PRINCIPAL_TABLE_FULL The table space reserved for PRINCIPAL during KDCDEF generation is already occupied or no table spaces were reserved for PRINCIPAL.
B B	KC_SC_PRINCIPAL_TOO_LONG The principal is longer than the value specified in MAX PRINCIPAL-LTH.
	KC_SC_INVALID_CLUSTER_RESTART Only for UTM cluster applications: Invalid value for *restart*.

11.2.4 KC_CREATE_STATEMENTS - Create KDCDEF control statements (inverse KDCDEF)

KC_CREATE_STATEMENTS allows you to start an inverse KDCDEF run during the application run (online). The inverse KDCDEF creates KDCDEF control statements from the configuration data. In this way, all changes resulting from dynamically adding, modify and deleting objects can be carried over to a new generation.

The KDCDEF control statements created by the inverse KDCDEF represent a consistent state of the configuration of the running application in the following sense:

The changes to the configuration data carried out by a transaction are always taken fully into account by an inverse KDCDEF running simultaneously.

See also the section on inverse KDCDEF runs in the openUTM manual “Generating Applications”.

- The inverse KDCDEF allows you to create the following KDCDEF control statements:
- CON statements for transport connections to remote LU6.1 applications
- KSET statements for all key sets
- LSES statements for all LU6.1 sessions
- LTAC statements for transaction codes by means of which service programs are started in partner applications.
- LTERM statements for all LTERM partners that do not belong to an LTERM pool or a multiplex connection
- PTERM statements for all clients and printers that have been explicitly added to the configuration
- PROGRAM statements for all program units and VORGANG exits
- TAC statements for all transaction codes and TAQ queues in the application
- USER statements for all user IDs including their queues

The inverse KDCDEF creates a control statement for each object of the specified type that is contained in the configuration, irrespective of whether these objects were loaded dynamically or not and whether their properties have been modified or not. The inverse KDCDEF does not create control statements for objects deleted with KC_DELETE_OBJECT.

You can find detailed information on the inverse KDCDEF in chapter “Generating KDCDEF statements from the KDCFILE” on page 101.

Controlling the inverse KDCDEF run

The inverse KDCDEF differentiates between the following seven object groups

First group	LTERM partners, clients, printers (object types: KC_LTERM, KC_PTERM)
Second group	program units, transaction codes, TAC queues (object types: KC_PROGRAM, KC_TAC)
Third group	user IDs (object type: KC_USER)
Fourth group	key sets (object type: KC_KSET)
Fifth group	transaction codes via which the service programs are started in partner applications (object type: KC_LTAC)
Sixth group	transport connections to LU6.1 applications (object type: KC_CON)
Seventh group	LU6.1 sessions (object type: KC_LSES)

You can use the KC_CREATE_STATEMENTS call to create KDCDEF control statements for objects of one or more of these groups.

You must specify the file in which UTM is to write the KDCDEF control statements in the KC_CREATE_STATEMENTS call. You can have all control statements written into one file or you can specify a file for each of the object groups. You may also specify in the call whether UTM is to create a new file or append the data to an existing file.

B B On BS2000 systems, the control statements can also be written to an LMS library element instead of a file. The procedure for library elements is similar to the procedure for files.

Execution of an inverse KDCDEF run

The time at which the inverse KDCDEF run is started and execution itself are dependent on the current state of the application. The following two cases can occur:

- The inverse KDCDEF run is started asynchronously if transactions that have write access to the configuration data of the objects are running at the time of the KC_CREATE_STATEMENTS call. The inverse KDCDEF run is only started after these transactions have been completed. In the case of new transactions that are intended to change data in the object tables, the corresponding calls to change the configuration data of the application are rejected until the inverse KDCDEF run is completed (i.e. until the asynchronous job is processed).

  The following also applies in UTM cluster applications:
  In all running node applications, an administration action which applies globally to the cluster results in this type of transaction which may delay the start on the inverse KDCDEF. Conversely, the execution of a global administration action at a running node may be delayed if an inverse KDCDEF is currently running there.

- The inverse KDCDEF run is started synchronously if **no** transactions that have write access to the configuration data of the objects are running at the time of the KC_CREATE_STATEMENTS call. The run is already finished when control returns to the administration program. This means that, at this point in time, all of the KDCDEF control statements requested have been created and written to the files specified.

Results of the inverse KDCDEF runs

After a successful inverse KDCDEF run, the control statements requested are stored in the files specified in the call. These files can be used as input for the UTM generation tool KDCDEF when regenerating the application. You must pass each of the files to KDCDEF with the KDCDEF control statement OPTION DATA=filename. The files can be edited and modified.

B B B The same applies if the control statements on BS2000 systems are written to LMS library elements instead of to files. However, whether or not elements can be edited depends on their type: only text-type elements can be modified.

Transaction management / cluster

The KC_CREATE_STATEMENTS call only reads the data in the KDCFILE. For this reason, the call is not subject to transaction management. The call cannot be undone in the same transaction using an RSET call.

The following applies in UTM cluster applications:
The call applies locally to the node, i.e. an inverse KDCDEF run for the generation of control statements from the configuration data is only started in this node application. It is sufficient for the effect to be local to the node since the same objects exist in every node application. An effect global to the cluster would simply generate identical KDCDEF statements.

If node applications with different generations are running (during an online update), then the call is rejected since the result would otherwise depend on the application at which the call was executed.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area	**identification area**	**selection area**	**data area**
Create KDCDEF control statements online	Operation code: KC_CREATE_STATEMENTS	——	——	Data structure with information on the type of control statements to be created as well as the names and write modes of the files

Parameter settings

Parameter area

1.

Field name	Contents
version	KC_ADMI_VERSION_1
retcode	KC_RC_NIL
version_data	KC_VERSION_DATA
opcode	KC_CREATE_STATEMENTS
id_lth	0
select_lth	0
data_lth	Length of data in the data area

Identification area

—

Selection area

—

Data area

2.

Data structure kc_create_statements_str

KDCADMI call

```
KDCADMI (&parameter_area, NULL, NULL, &data_area)
```

Data returned by UTM

Parameter area

	Field name	Contents
3.	retcode	Return codes

1. In *data_lth* you specify the length of the data structure *kc_create_statements_str*.
2. In the data area you must specify whether or not UTM is to create the KDCDEF control statements for each of the object groups. If UTM is to create control statements for an object group, you must also specify the file in which UTM is to write the control statements and the write mode of the file. The header file *kcadminc.h* contains the following data structure definition for passing information to UTM.

   Definition of constants

```
#define KC_FILE_NAME_LTH    54
#define KC_ELEM_NAME_LTH    64
#define KC_VERSION_LTH      24
#define KC_TYPE_LTH          8
```

   Definition of the index constant

```
typedef enum
{  KC_DEVICE_STMT      = 0,
   KC_PROGRAM_STMT     = 1,
   KC_USER_STMT        = 2,
   KC_KSET_STMT        = 3,
   KC_LTAC_STMT        = 4,
   KC_CON_STMT         = 5,
   KC_LSES_STMT        = 6,
   KC_MAX_STMT_TYPE    = 6,
   KC_DUMMY_STMT_TYPE  = 7
} KC_INVDEF_TYPE;
```

   Definition of the data structure

```
struct kc_create_statements_str
{  struct
   {  char  create_control_stmts;
      char  file_name[KC_FILE_NAME_LTH];
      char  file_mode;
      char  lib_name[KC_FILE_NAME_LTH];
      char  elem_name[KC_ELEM_NAME_LTH];
      char  vers[KC_VERSION_LTH];
      char  type[KC_TYPE_LTH];
   } type_list[(int)KC_MAX_STMT_TYPE + 1];
```

```
  char stmt_type;
  char file_error_code[4];
};
```

The KC_INVDEF_TYPE index of the *type_list* array specifies the group to which the objects belong:

KC_DEVICE_STMT
: stands for the first group, consisting of the LTERM partners, clients and printers. The KDCDEF control statements LTERM and PTERM are created in this group.

KC_PROGRAM_STMT
: stands for the second group, consisting of the program units, transaction codes and TAC queues. The KDCDEF control statements PROGAM and TAC are created in this group.

KC_USER_STMT
: stands for the third group, consisting of the UTM user IDs. The KDCDEF USER control statements are created in this group.

KC_KSET_STMT
: Stands for the 4th group, the KSETs. The KDCDEF control statements KSET are generated in this group.

KC_LTAC_STMT
: stands for the 5th group, the transaction codes by means of which service programs are started in partner applications. The KDCDEF LTAC control statements are created in this group.

KC_CON_STMT
: Stands for the 4th group, the transport connections to LU6.1 applications. The KDCDEF control statements CON are generated in this group.

KC_LSES_STMT
: stands for the 7th group, the LU6.1 sessions. The KDCDEF LSES control statements are created in this group.

The fields in the data structures must be supplied with the following data:

create_control_stmts
: You specify here whether or not KDCDEF control statements are to be created for the object group belonging to KC_INVDEF_TYPE.

  ´Y´ KDCDEF control statements are to be created for this object group.

  ´N´ No KDCDEF control statements are to be created for this object group. You can also specify the null byte (´\0´) in place of the ´N´.

file_name The name of the file in which the KDCDEF control statements are to be written. The name may be up to 54 characters long. It must conform to the file naming conventions of the operating system under which the application is running.

X/W X/W X/W X/W On Unix, Linux and Windows systems, the file name can be specified as an absolute or relative path name. A relative file name specification will write the KDCDEF control statements to a file in the directory in which the application was started.

file_mode Write mode of the file in *file_name* bor of the element in *elem_name*

´C´ Create:
UTM is to create a new file with the name *file_name* or a new element with the name *elem_name*.

B B B B B B B On BS2000 systems, inverse KDCDEF generates an SAM file or an LMS library element. Here, the following applies:
– If a file of the same name already exists then it must be a SAM file. The existing SAM file is then overwritten.
– If an element of the same name already exists and if *HIGHEST-EXISTING or *UPPER-LIMIT is specified for vers=C'<version> then an existing element of the specified version is overwritten.

´E´ Extend:
UTM is to append the KDCDEF control statements to an existing file or to an existing element.

– If the file with the name *file_name* does not exist, UTM will create it.

B B B B – If an LMS library is specified in *lib_name* on BS2000 systems then the library must already exist. In this case, an existing element of the specified version is extended. If the element does not yet exist in this version then it is created.

B B B lib_name Name of the LMS library in which the KDCDEF control statements are to be stored. The name can be up to 54 characters in length. It must comply with the conventions for file names on the BS2000 system.

B B If the name is shorter than the field length then it must be padded with spaces.

B It is not permissible to specify *file_name* and *lib_name* at the same time.

B B If *lib_name* is specified then it is also necessary to enter values for *elem_name*, *vers* and *type*.

elem_name

Name of the LMS library element to which the KDCDEF control statements are to be written. The name can be up to 64 characters in length. If the name is shorter than the field length then it must be padded with spaces. The name must comply with the conventions for LMS element names

vers Version of the LMS library element to which the KDCDEF control statements are to be written. The version can be up to 24 characters in length and must comply with the conventions for LMS version specifications. If the version is shorter than the field length then it must be padded with spaces.

You can also enter the following character strings as the version:

*HIGHEST-EXISTING

The statements are written to the highest version of the specified element present in the library.

*UPPER-LIMIT

The statements are written to the highest possible version of the specified element. LMS indicates this version by means of an "@".

*INCREMENT

A new version is created for the specified element. *INCREMENT may only be specified if *file_mode*=´C´.

These character strings may not be truncated!

type Type of the LMS library element to which the KDCDEF control statements are to be written. The type can be up to 8 characters in length and must comply with the conventions for LMS type specifications. If the type is shorter than the field length then it must be padded with spaces.

It is recommended to use the LMS type "S" for *type*.

KDCDEF does not check whether the specifications in *elem_name*, *vers* or *type* comply with the LMS syntax rules. For further information on the syntax rules for the names of LMS elements and a specification of version and type, see the manual "LMS SDF Format".

stmt_type If a value other than KC_MC_OK is returned as the main code then the field *stmt_type* contains the index from KC_INVDEF_TYPE, to which the error message refers.

file_error_code

If the subcode KC_SC_FILE_ERROR is returned when an error occurs then the field *file_error_code* contains the DMS error code or (on BS2000 systems) the associated PLAM error code.

The *type_list* array is processed in order starting with the first array element (index KC_DEVICE_STMT) and proceeding to the last array element (index KC_LSES_STMT) when UTM is called.

If UTM is to create KDCDEF control statements for all three object groups, then the *create_control_stmts* field must be set to ´Y´, the *file_name* field must be set to the file name and the *file_mode* field must be set to the write mode of the file in each array element.

If all of the control statements are to be written to one file, then you should ensure that the correct write mode has been set.
You can set the write mode to ´C´ or ´E´ for the first entry of the file or the LMS library element. In the following array elements, however, the write mode must be set to ´E´. Otherwise, the control statements just created will be overwritten.

If UTM is not to create control statements for one of the object groups, then *create_control_stmts*=´N´ (or nothing at all) is to be specified in the corresponding array element.

3. In the *retcode* field UTM outputs the return codes of the call. In addition to the codes listed in section “Return codes” on page 174, the following return codes can also arise:

Main code = KC_MC_OK The call was processed without errors. **Subcode:**
KC_SC_ASYN_INIT The job was accepted; the inverse KDCDEF will be started asynchronously as soon as all transactions that modify configuration data have terminated.

Main code = KC_MC_DATA_INVALID Invalid or missing data in the data area. **Subcodes:**
KC_SC_DATA_MISSING No data was specified in the data structure passed in the data area.
KC_SC_NO_INFO Invalid data was specified in the data structure passed in the data area.
BB KC_SC_FILE_LIBRARY_MISMATCH Both a file name (*file_name*) and an LMS library (*lib_name*) have been specified.
BB KC_SC_LMS_ELEMENT_MISSING An LMS library (*lib_name*) was specified but no element name (*elem_name).*
BB KC_SC_LMS_VERSION_MISSING An LMS library (*lib_name*) was specified but no element version (*vers*).

Main code = KC_MC_DATA_INVALID Invalid or missing data in the data area. **Subcodes:**
B B KC_SC_LMS_TYPE_MISSING An LMS library (*lib_name*) was specified but no element type (*type*).
B B KC_SC_LMS_VERSION_MODE_MISMATCH *INCREMENT was specified as LMS version but *file_mode* is not ´C´.

Main code = KC_MC_MEMORY_INSUFF There is not enough internal UTM memory available. **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the present time. **Subcode:**
KC_SC_INVDEF_RUNNING An inverse KDCDEF is currently running or an inverse KDCDEF run is being prepared asynchronously, i.e. the job cannot be processed at the present time.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_GEN KDCDEF control statements are to be generated for objects whose types were not generated, such as USER commands for an application without user IDs.
KC_SC_FILE_ERROR One of the files in which the KDCDEF control statements are to be written cannot be written to. A DMS error code or (on BS2000 systems) a PLAM error code is returned in the field *file_error_code*. This code provides ingormation about the error that has occurred
KC_SC_NO_INFO The page pool used to temporarily store the parameters passed is full.
KC_SC_CLUSTER_CONF_INCONS Only for UTM cluster applications: The running node applications have different generations.

11.2.5 KC_DELETE_OBJECT - Delete objects

KC_DELETE_OBJECT allows you to delete objects belonging to one of the following object types from the configuration:

– transport connections to remote LU6.1 applications (KC_CON)
– key sets (KC_KSET)
– LU6.1 sessions (KC_LSES)
– transaction codes by means of which service programs can be started in partner applications (KC_LTAC)
– LTERM partners used to connect clients and printers (KC_LTERM)
– clients and printers (KC_PTERM)
– application program units and VORGANG exits (KC_PROGRAM)
– transaction codes and TAC queues (KC_TAC)
– user IDs including their queues (KC_USER)

You can find more detailed information on dynamically deleting objects from the configuration in chapter “Changing the configuration dynamically” on page 67.

Objects that you are not allowed to delete

– LTERM partners that belong to an LTERM pool or multiplex connection
– LTERM partners belonging to an LTERM group (group or primary LTERM) or to an LTERM bundle (master or slave LTERM),
– printer control LTERMs
– the LTERM partner KDCMSGLT that UTM creates internally for the MSGTAC service
– program units that belong to the START, SHUT, FORMAT or INPUT event exits
– program units and VORGANG exits that are statically linked into the application program
– the KDCMSGTC, KDCSGNTC, KDCBADTC transaction codes of the event services
– transaction codes assigned to a transport system access point (BCAMAPPL) as SIGNON-TAC
– the dead letter queue KDCDLETQ,
– statically linked programs with event exits
– the KDCSHUT administration command of the KDCADM administration program
– the KDCTXCOM and KDCTXRLB transaction codes reserved for XATMI
– the KDCMSGUS user ID that UTM creates internally for the MSGTAC service.
– a user ID assigned to a terminal for automatic KDSIGN or to a UPIC, APPLI or SOCKET client as a connection user ID

The following must be observed when deleting objects:

- A program unit or a VORGANG exit may only be deleted after all the transaction codes belonging to them have been deleted.
- An LTERM partner may only be deleted if no more clients or printers are assigned to it.
- A user ID may only be deleted if there are no more users or clients signed on under this user ID, i.e.:
  - The user must not be signed on in a standalone application with SIGNON MULTI-SIGNON=NO.
  - In a standalone application with SIGNON MULTI-SIGNON=YES,
    - a user with RESTART=YES must not be signed on,
    - a user with RESTART=NO must not be signed on via a terminal connection.
  - In a UTM cluster application with SIGNON MULTI-SIGNON=NO,
    - no genuine user may be signed on,
    - a connection user must not be signed on at the node application at which the administration 'Delete' call is executed.
  - In a cluster application with SIGNON MULTI-SIGNON=YES,
    - no genuine user with RESTART=YES may be signed on,
    - a connection user must not be signed on at the node application at which the administration 'Delete' call is executed,
    - a user with RESTART=NO may not be signed on via a terminal connection at the node application at which the administration 'Delete' call is executed.
- When a client/printer is deleted, it must not be connected to the application.
- A logical connection for distributed processing by means of LU6.1 may not be deleted when it is not set up.
- An LU6.1 session may only be deleted when it is not set up and is not in the P state (prepare to commit).

Effects of deletion during the application run

We distinguish two methods of deletion:

- immediate delete (with *subopcode1*=KC_IMMEDIATE).
  This method is only possible in conjunction with user IDs (KC_USER) and transport connections to LU6.1 applications (KC_CON). The immediate deletion of a user ID or a CON object causes the space in the object table to be freed up and made available for further use immediately. Immediate deletion is only possible for users IDs (KC_USER) and transport connections to LU6.1 applications (KC_CON). You can generate a new user ID using the same name directly after the deletion.
  Immediate deletion is only possible in standalone UTM applications.

- delayed delete (with *subopcode1*=KC_DELAY)
  Delayed deletion has the effect of a “permanent lock”. This process does not free up space in the object table.
  The object’s name remains reserved, i.e. you cannot generate dynamically a new object using this name within the same name class.
  The delayed deletion of transport connections to LU6.1 applications (KC_CON) is not possible in standalone UTM applications.

  In UTM cluster applications, only delayed deletion is possible.
  In UTM cluster applications, it is possible to delete objects with an update generation without having to terminate the entire UTM cluster application. To implement this change in all the running node applications, it is necessary to terminate the individual node applications one after the other and then start them with the new generation.

  For details see openUTM manual “Using openUTM Applications” subsection "Update generation in a cluster”.

The deletion of an object cannot be undone.

The inverse KDCDEF does not create KDCDEF control statements for deleted objects.

The effects of the deletion of an object on unprocessed asynchronous jobs, asynchronous messages, open dialog services etc. that relate to that object are described in chapter “Changing the configuration dynamically” on page 67.

Procedure / period of validity / transaction management / cluster

The call is subject to transaction management. The object is deleted from the configuration only after the program unit run has ended (for PEND). The call can be rolled back with an RSET call that is executed in the same transaction.

The deletion remains effective even after the UTM-S- and UTM-F applications have terminated; it cannot be undone.

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. objects are deleted from the configuration in all the node applications.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identification area**	**selection area**	**data area**
Delete transport connections to LU6.1 applications	*subopcode1*: KC_DELAY or KC_IMMEDIATE (see page 257) *obj_type*: KC_CON	Name of the partner application, name of the computer, name of the local application	——	——
Delete a key set	*subopcode1*: KC_DELAY *obj_type*: KC_KSET	Name of the key set	——	——
Delete an LU6.1 session	*subopcode1*: KC_DELAY *obj_type*: KC_LSES	Local half-session name	——	——
Delete a transaction code by means of which service programs are started in partner applications	*subopcode1*: KC_DELAY *obj_type*: KC_LTAC	Name of the transaction code	——	——
Delete an LTERM partner from the configuration	*subopcode1*: KC_DELAY *obj_type*: KC_LTERM	Name of the LTERM partner	——	——
Delete a client or printer from the configuration	*subopcode1*: KC_DELAY *obj_type*: KC_PTERM	Name of the client/printer, computer name, BCAMAPPL name	——	——
Delete a program unit from the configuration	*subopcode1*: KC_DELAY *obj_type*: KC_PROGRAM	Program name	——	——
Delete a transaction code or TAC queue from the configuration	*subopcode1*: KC_DELAY *obj_type*: KC_TAC	TAC name	——	——
Delete a user ID including its queue from the configuration	*subopcode1*: KC_DELAY or KC_IMMEDIATE (see page 257) *obj_type*: KC_USER	User ID	——	——

[1] The operation code KC_DELETE_OBJECT must be specified in the parameter area in all cases.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_DELETE_OBJECT
1.	subopcode1	KC_DELAY / KC_IMMEDIATE
2.	obj_type	Object type
3.	obj_number	1
4.	id_lth	Length of the object name in the identification area
	select_lth	0
	data_lth	0

Identification area

5.	Object name

Selection area

—

Data area

—

KDCADMI call

```
KDCADMI (&parameter_area, &identification_area, NULL, NULL)
```

Data returned by UTM

Parameter area

	Field name	Contents
6.	retcode	Return codes

1. In *subopcode1* you specify the method of deletion.

   KC_DELAY
   : if an object is to be marked as deleted, i.e. it is to be permanently locked (delayed delete).
   KC_DELAY in *obj_type*=KC_CON is not permitted in standalone openUTM applications.

   KC_IMMEDIATE
   : is only allowed in standalone openUTM applications with *obj_type*=KC_USER and *obj_type*=KC_CON.
   You must specify KC_IMMEDIATE, if a user ID or transport connection is to be deleted immediately.

2. In the *obj_type* field you must specify the type of object to be deleted.
   You can specify the following object types:

   KC_CON, KC_KSET, KC_LSES, KC_LTAC, KC_LTERM, KC_PROGRAM, KC_PTERM, KC_TAC (transaction code including TAC queue) and KC_USER (user ID including associated queue)

3. Only one object can be deleted per call. For this reason, *obj_number* = 1 must be specified.

4. In the *id_lth* field you must specify the length of the object name that you are passing in the identification area to UTM.

5. In the identification area you must pass the name of the object to be deleted. The full name of the object must be specified. You must enter the following data:

   for *obj_type*=KC_CON:
   in the data structure *kc_triple_str* in the union *kc_id_area*; the name of the partner application, the name of the computer on which the application can be found and the name of the local application (BCAMAPPL name of the CON)

   for *obj_type*=KC_KSET:
   the name of the key set (*kc_name8* in the union *kc_id_area*)

   for *obj_type*=KC_LSES:
   the name of the local half session (*kc_name8* in the union *kc_id_area*)

   for *obj_type*=KC_LTAC:
   the name of the transaction code by means of which remote service programs are started (*kc_name8* in the Union *kc_id_area*)

   for *obj_type*=KC_PTERM:
   in the data structure *kc_triple_str* in the union *kc_id_area;* the name of the client/printer, the name of the computer on which it can be found and the name of the UTM application (of the BCAMAPPL) through which the connection is to be established

for *obj_type*=KC_PROGRAM:
the name of the program unit (*kc_name32* in the union *kc_id_area)*

for *obj_type*=KC_TAC:
the name of the transaction code or the TAC queue (*kc_name8* in the union *kc_id_area)*

for *obj_type*=KC_USER:
the name of the user ID (*kc_name8* in the union *kc_id_area)*

6. In the *retcode* field UTM outputs the return codes of the call. In addition to the return codes listed in section “Return codes” on page 174, the following codes can also be returned. Some of these return codes may arise independently of the object type specified; others only occur for certain object types.

Type-independent return codes:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_INVALID_OBJECT The object specified does not exist.
KC_SC_DELETE_NOT_ALLOWED The object cannot be deleted, it has already been deleted or it has just been created (in the same transaction).
KC_SC_JCTL_RT_CODE_NOT_OK Only in UTM cluster applications: An internal UTM error occurred during the global deletion of a object. Please contact system support.
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration changes are possible since the generation of the node applications is not consistent at present.
KC_SC_GLOB_CRE_DEL_LOCKED Only in UTM cluster applications: It is not possible to delete an object at present because the generation or deletion of an object or the generation, deletion or activation of an RSA key pair has not yet been completed in a node application.

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the present time. **Subcode:**
KC_SC_INVDEF_RUNNING An inverse KDCDEF is running, i.e. the job cannot be processed at the present time.

Main code = KC_MC_RECBUF_FULL The buffer containing the restart information is full. (See the openUTM manual “Generating Applications”, KDCDEF control statement MAX, parameter RECBUF). **Subcode:**
KC_SC_NO_INFO

Return codes for obj_type = KC_CON:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CONNECTED The specified transport connection cannot be deleted because it is currently in operation.

Maincode = KC_MC_PAR_INVALID An invalid value has been entered or a field has not been set in the parameter area. **Subcode:**
KC_SC_SUBOPCODE1 Only in UTM cluster applications: The specified transport connection cannot be deleted, deletion with subcode KC_IMMEDIATE not permitted.

Return codes for obj_type = KC_KSET:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_KSET_NOT_ADMINISTRABLE The KDCAPLKS key set cannot be deleted.

Return codes for obj_type = KC_LSES:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CONNECTED The connection with the specified session name cannot be deleted because it is currently set up.
KC_SC_PTC_STATE The session has the transaction status P (prepare to commit). When it has this status it cannot be deleted.

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED The session is currently occupied (not active).

Return codes for obj_type = KC_LTERM:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_LTERM_IS_MASTER The LTERM partner cannot be deleted because it is the master of an LTERM bundle.
KC_SC_LT_DEL_GROUP_MASTER The LTERM partner cannot be deleted because it is the primary LTERM of an LTERM group.
KC_SC_LT_DEL_SLAVE The LTERM partner cannot be deleted because it is the slave of an LTERM bundle.
KC_SC_LT_DEL_ALIAS The LTERM partner cannot be deleted because it is the group LTERM of an LTERM group.
KC_SC_REF_PTERM_NOT_DELETED The LTERM partner cannot be deleted because a client/printer assigned to the LTERM partner has not yet been deleted.
KC_SC_LTERM_IS_CTERM The LTERM partner specified is a printer control LTERM. It cannot be deleted.
KC_SC_OBJECT_TYPE_NOT_LTERM The object specified cannot be deleted because: – it is an LTERM partner that belongs to an LTERM pool or multiplex connection – the name specified belongs to an LPAP or OSI-LPAP partner.
KC_SC_LTERM_NOT_ADMINISTRABLE The LTERM partner specified cannot be administered (for example, the LTERM partner KDCMSGLT which is created internally by UTM for the event service MSGTAC).

Return codes for obj_type = KC_PROGRAM:

Main code = KC_MC_REJECTED The call was rejected by UTM. The object cannot be deleted. **Subcodes:**
KC_SC_REF_TAC_NOT_DELETED A transaction code belonging to the program unit specified has not yet been deleted.

Main code = KC_MC_REJECTED The call was rejected by UTM. The object cannot be deleted. **Subcodes:**
KC_SC_PROGRAM_IS_STATIC The program unit cannot be deleted from the configuration because it belongs to a load module with load mode STATIC.
KC_SC_PROGRAM_IS_USER_EXIT The object specified is an event exit that was statically configured with the KDCDEF control statement EXIT (START, SHUT, FORMAT or INPUT exit).

Return codes for obj_type = KC_PTERM:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_PTERM_CONNECTED The client/printer specified cannot be deleted because it is currently connected to the application.
KC_SC_OBJECT_TYPE_NOT_PTERM The object specified cannot be deleted because: – it is a client that is connected to the application through an LTERM pool, i.e. that was not configured explicitly – the specified name was created during KDCDEF generation with a MUX statement (multiplex connection) – the name specified belongs to an object that was configured for distributed processing through OSI TP or LU6.1.

Return codes for obj_type = KC_TAC:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_TAC_NOT_ADMINISTRABLE The transaction code specified cannot be administered (KDCMSGTC, KDCBADTC, KDCSGNTC, for example) or cannot be deleted (the transaction code KDCSHUT and the Dead Letter Queue).
KC_SC_DELETE_NOT_ALLOWED The specified transaction code cannot be deleted (for example, a transaction code assigned to a transport access point as SIGNON-TAC)

Return codes for obj_type = KC_USER (*subopcode1* = KC_DELAY or KC_IMMEDIATE):

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_USER_CONNECTED A client/user with the user ID specified is currently signed on to the application.
KC_SC_APPLICATION_WITHOUT_USER The application was generated without user IDs.
KC_SC_USER_NOT_ADMINISTRABLE The user ID cannot be administered because it is, for example, the user ID KDCMSGUS that UTM creates internally for the MSGTAC event service.
KC_SC_AUTO_SIGN_USER The user ID cannot be deleted, because it is assigned to an LTERM partner for automatic KDSIGN or as a connection user ID.

obj_type = KC_USER and *subopcode1* = KC_IMMEDIATE:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_ASYN_SERVICE_RUNNING The user ID currently cannot be deleted because there is still an asynchronous service running under this user ID.
KC_SC_CLIENT_SIGNED Immediate deletion of the user ID is currently not possible because a UPIC client, TS application or OSI TP partner is still signed on with this user ID.
KC_SC_DEADLOCK Deadlock locking user-specific long-term storage (ULS)
KC_SC_TIMEOUT Timeout locking user-specific long-term storage (ULS)
KC_SC_OWNER_IN_TA User-specific long-term storage (ULS) cannot be locked because it is disabled by a transaction in which one of the KDCS calls PEND KP or PGWT KP was issued.
KC_SC_PTC_STATE There is a transaction in the state PTC (prepare to commit) for the user ID.
KC_SC_BOTTLENECK Services are stacked for the user ID, and a memory bottleneck has occurred.
KC_SC_ALREADY_LOCKED The assigned ULS is locked by another transaction.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ENOUGH_TASKS The UTM application does not currently have enough free processes to be able to wait for the lock of user-specific long-term storage (ULS) locked by a PTC transaction. Attempt to delete the user again later.

Maincode = KC_MC_PAR_INVALID An invalid value has been entered or a field has not been set in the parameter area. **Subcode:**
KC_SC_SUBOPCODE1 Only in UTM cluster applications: Deletion with subcode KC_IMMEDIATE is not permitted.

11.2.6 KC_ENCRYPT - Create, delete, read RSA key pairs

With KC_ENCRYPT, you can create a new application's RSA key pair, replace an application's RSA key pair by a new pair, delete an RSA key pair or read the public key of an RSA key pair.

B
B
B
B
B

UTM applications on BS2000 systems also support encryption for connections with some terminal emulations. However, these connections do not use the openUTM RSA key pair. Instead, a key pair generated by VTSU-B is employed. Consequently, changing the RSA key pair of openUTM has no effect whatsoever on encryption using VTSU-B.

Prerequisites

You can only use the encryption functions, if the add-on openUTM-CRYPT is installed (see release information on openUTM).

Encryption methods

openUTM offers encryption functions for passwords and user data (messages), in order to improve the security for connections between UTM server applications and UPIC clients.

For encryption purposes openUTM uses a combination of the AES method (Advanced Encryption Standard) and the RSA method (named after Rivest, Shamir and Adleman). The DES method (Data Encryption Standard) can also be used for partners who do not yet support the AES method.

For transferring on a connection, user data and passwords are encrypted with an AES or DES key. Client and UTM application use the same AES/DES key to encrypt and decrypt messages. This key is generated by the client and is transferred to the UTM application when a connection is established. The key is connection-specific, i.e. for each connection it is created a individual key which is used only for this connection only.

To increase the level of security, the AES or DES key itself is transferred in encrypted form. For this purpose, one or more RSA key pairs are created for the UTM application at generation. An RSA key pair consists of a public and a private key. The public key is transferred to the client by the openUTM application as soon as the connection is set up. The client uses it to encrypt the AES or DES key. To decrypt this key, the UTM application uses the private key which is known only to the UTM application.

Several encryption levels are possible and a separate RSA key pair with a specific key length is used for each level. The longest available RSA key is always exchanged for clients whose encryption level was not defined at generation. If this key is not needed, it should be deleted for performance reasons.

You will find further information on encryption in the openUTM manuals “Concepts and Functions” and “Generating Applications”.

Functional scope of KC_ENCRYPT

An RSA key pair that is valid for a specific encryption level is used for all encrypted client connections that use this encryption level. For reasons of security, you should therefore replace the RSA key pairs of your UTM application by new key pairs at regular intervals.

For this purpose, KC_ENCRYPT offers the following functions:

- Create a new RSA key pair

  KC_ENCRYPT with *subopcode1*=KC_CREATE_KEY makes UTM generate a new RSA key pair. However, UTM does not use this new key pair for encryption, before you activate it by dispatching a further KC_ENCRYPT call (with *subopcode1*=KC_ACTIVATE_KEY).

  You cannot create a new key pair unless the key pair last created with the same encryption level has already been activated with *subopcode1*=KC_ACTIVATE_KEY or has been deleted with *subopcode1*=KC_DELETE_KEY, i.e. there must be no not yet activated key pair of the same encryption level for the application.

- Delete a key pair

  You use KC_ENCRYPT with *subopcode1*=KC_DELETE_KEY to delete a key pair that has not yet been activated. You use KC_ENCRYPT with *subopcode1* = KC_DELETE_ACTIVE_KEY to delete an activated key pair.

  You can delete activated key pairs of encryption levels 3 and 4 only. Activated key pairs of encryption levels 1 and 2 are always needed by openUTM.

- Activate a previously created RSA key pair

  KC_ENCRYPT with *subopcode1*=KC_ACTIVATE_KEY causes an RSA key pair currently being used to be replaced by a RSA key pair created using KC_ENCRYPT, i.e. the next time a connection is established to an appropriately generated client, the public key of the new RSA key pair is transmitted to the client.

- Read a public key

  You can read the public key of an RSA key pair that was last created and that is not activated yet using KC_ENCRYPT *subopcode1*=KC_READ_NEW_PUBLIC_KEY. KC_ENCRYPT *subopcode1*=KC_READ_ACTIV_PUBLIC_KEY allows you to read the public key of an currently active RSA key pair.

  This function gives you added possibilities of increasing data security on your connection:

  In order for a client to be able to verify whether the public key received via the connection to the UTM application actually truly comes from that UTM application, you should read the public key, transfer it to the client using the appropriate procedure and deposit it there.

When the UTM application transmits the public key to the client the next time a connection is established, the client can compare the transmitted key with the one already stored.

It is therefore a good idea to transmit the public key of a newly created RSA key pair to all clients involved, i.e. all clients that support message encryption.

Transaction management / duration of effectiveness / cluster

Creating, activating and deleting a RSA key pair is subject to transaction management. You can create or activate a new key pair within a transaction. A new public key can only be read after the transaction is terminated.

The RSA key pair remains active until a new pair is created and activated or until the application is regenerated. In the event of regeneration, UTM automatically generates a new RSA key pair if the OPTION GEN-RSA-KEYS=YES statement is specified for the KDCDEF run (default setting).

Reading the public key is not subject to transaction management.

The effect of the call persists beyond the current application run.

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e.

– if you use the KC_ENCRYPT function to generate a new key pair at a node application then this key pair is also distributed to the other node applications so that all the node applications possess the same key pairs.
– if you activate or delete a previously generated key pair at a node application then this action is replicated at all the other node applications.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identifica-tions area**	**selection area**	**data area**
Create RSA key pair	*subopcode1*: KC_CREATE_KEY *subopcode2*: encryption level	——	——	——
Delete non-activated RSA key pair	*subopcode1*: KC_DELETE_KEY *subopcode2*: encryption level	——	——	——
Delete activated RSA key pair	*subopcode1*: KC_DELETE_ACTIVE_KEY *subopcode2*: encryption level	——	——	——
Activate RSA key pair	*subopcode1*: KC_ACTIVATE_KEY *subopcode2*: encryption level	——	——	——
Public key of a not yet activated RSA key pair	*subopcode1*: KC_READ_NEW_PUBLIC_KEY *subopcode2*: encryption level	——	——	Pointer to a data area into which UTM can return the public key.
Public key of the currently active RSA key pair	*subopcode1*: KC_READ_ACTIV_PUBLIC_KEY *subopcode2*: encryption level	——	——	Pointer to a data area into which UTM can return the public key.

[1] In all cases, the operation code KC_ENCRYPT must be specified in the parameter area.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_ENCRYPT
1.	subopcode1	KC_CREATE_KEY / KC_ACTIVATE_KEY / KC_DELETE_KEY/ KC_DELETE_ACTIVE_KEY / KC_READ_NEW_PUBLIC_KEY / KC_READ_ACTIV_PUBLIC_KEY
2.	subopcode2	KC_NO_SUBOPCODE / KC_ENC_LEV_1 / KC_ENC_LEV_2 / KC_ENC_LEV_3 / KC_ENC_LEV_4 /
	obj_number	0
	id_lth	0
	select_lth	0
3.	data_lth	length of data area / 0

Identification area

—

Selection area

—

Data area

—

KDCADMI call

KDCADMI (&parameter_area, NULL, NULL, NULL) or
KDCADMI (&parameter_area, NULL, NULL, &data_area)

Data returned by UTM

Parameter area

	Field name	Field contents
4.	retcode	return code
5.	data_lth_ret	length of data returned/ 0

Data area

6.	Data structure kc_encrypt_str / kc_encrypt_advanced_str / —

1. In the field *subopcode1*, you must specify which action UTM is to execute. You can enter the following subcodes:

   KC_CREATE_KEY
   : Generates a new RSA key pair.

   KC_ACTIVATE_KEY
   : Activates a RSA key pair created with KC_ENCRYPT.

   KC_DELETE_KEY
   : Deletes a not yet activated RSA key pair.

   KC_DELETE_ACTIVE_KEY
   : An activated RSA key pair is to be deleted. Only activated keys of encryption levels 3 and 4 can be deleted.
   This function is permitted only if the key pair has not been used by any object before deletion. It can be used, for example, after application regeneration and a subsequent KDCUPD to delete RSA keys that are no longer needed in the newly generated application.

   KC_READ_NEW_PUBLIC_KEY
   : Reads the public key of a previously created and not yet activated RSA key pair.

   KC_READ_ACTIV_PUBLIC_KEY
   : Reads the public key of the active RSA key pair.

2. In the field *subopcode2*, you must indicate to which encryption level the action specified in *subopcode1* applies:

   KC_ENC_LEV_1
   : The action applies for keys of encryption level 1 (DES encryption and RSA key pair with a key length of 200 bits).

   KC_ENC_LEV_2
   : The action applies for keys of encryption level 2 (AES encryption and RSA key pair with a key length of 512 bits).

   KC_ENC_LEV_3
   : The action applies for keys of encryption level 3 (AES encryption and RSA key pair with a key length of 1024 bits).

   KC_ENC_LEV_4
   : The action applies for keys of encryption level 4 (AES encryption and RSA key pair with a key length of 2048 bits.

   KC_NO_SUBOPCODE
   : This value is supported to ensure source compatibility for existing clients. KC_NO_SUBOPCODE has the same effect as KC_ENC_LEV_1. However, different data structures are used when reading, see page 276.

3. In the field *data_lth*, you enter the following:

   – with *subopcode1*=KC_CREATE_KEY, KC_DELETE_KEY, KC_DELETE_ACTIVE_KEY or KC_ACTIVATE_KEY:
   data_lth=0. When you call KDCADMI, you should pass the zero pointer to UTM for *&data_area*.

   – with *subopcode1*=KC_READ_NEW_PUBLIC_KEY or KC_READ_ACTIV_PUBLIC_KEY:
   Length of the data area to which UTM is to return the public key of the RSA key pair. This data area must have the length of data structure *kc_encrypt_advanced_str*. For existing clients that work with *subopcode2*=KC_NO_SUBOPCODE, it must have the length of data structure *kc_encrypt_str*.
   When you call KDCADMI, you must pass the pointer to the data area to UTM.

4. In the field *retcode*, UTM supplies the return code of the call. Beside the return codes listed in section “Return codes” on page 174, one of the following return codes can also occur:

Maincode = KC_MC_REJECTED UTM rejected the call. **Subcode:**
KC_SC_NO_ENCRYPTION Encryption is not supported, because openUTM-CRYPT is not present.
KC_SC_NEW_KEY_ALREADY_EXISTS With *subopcode1*= KC_CREATE_KEY: A new key pair has already been generated for this encryption level.
KC_SC_NO_NEW_KEY_EXISTS With *subopcode1*=KC_READ_NEW_PUBLIC_KEY, KC_ACTIVATE_KEY, KC_DELETE_KEY: There is no new key for the specified encryption level.
KC_SC_NO_ACTIV_KEY_EXISTS With *subopcode1*= KC_READ_ACTIV_PUBLIC_KEY, KC_DELETE_ACTIVE_KEY: There is no activated key for the specified encryption level.
KC_SC_IN_USE_DEL_NOT_ALLOWED With *subopcode1*=KC_DELETE_ACTIVE_KEY: – The key pair for the specified encryption level may not be deleted because it is required by at least one object. – It is not permitted to delete a key pair of encryption level 1 or (these two key pairs are always needed by UTM).
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration changes are possible since the generation of the node applications is not consistent at present. Action: Please try again later.
KC_SC_GLOB_CRE_DEL_LOCKED Only in UTM cluster applications: It is not possible to generate, delete or activate an RSA key pair at present because the generation or deletion of an object or the generation, deletion or activation of an RSA key pair has not yet been completed in a node application. Action: Please try again later.

Maincode = KC_MC_RECBUF_FULL **Subcode:**
KC_SC_NO_INFO The buffer containing the restart information is full. (See openUTM manual “Generating Applications”, KDCDEF control statement MAX, parameter RECBUF)

Maincode = KC_MC_REJECTED_CURR The call cannot be processed at present. **Subcode:**
KC_SC_INVDEF_RUNNING Only in UTM cluster applications: An inverse KDCDEF is currently running, i.e. the job cannot be processed at present.

5. *data_lth_ret* contains the data length returned to the data area by UTM.
   - With *subopcode1*=KC_READ_NEW_PUBLIC_KEY and KC_READ_ACTIV_PUBLIC_KEY *data_lth_ret* ≠ 0.
     If the value in *data_lth_ret* is smaller than the data area available (*data_lth*), the contents of the data area is only defined in *data_lth_ret*.
   - In all other cases *data_lth_ret*=0
6. In the case where *subopcode1*=KC_READ_NEW_PUBLIC_ KEY or. KC_READ_ACTIV_PUBLIC_KEY, UTM returns the data structure *kc_encrypt_advanced_str* together with the public key of the specified encryption level. KC_READ_NEW_PUBLIC_KEY returns the key of the RSA key pair not yet activated. KC_READ_ACTIV_PUBLIC_KEY returns the key of the activated RSA key pair. UTM returns the data structure *kc_encrypt_str* for existing clients that work with *subopcode2*=KC_NO_SUBOPCODE.

   The data structure *kc_encrypt_advanced_str* is defined as follows:

struct kc_encrypt_advanced_str

`char buf_lth[4];`
`char en_buffer[2048];`
`char en_key_lth[4];`

   The fields of the data structure have the following meanings:

   buf_lth length of the data buffer *en_buffer* used.

   en_buffer contains the public key that was read.

   en_key_lth length of the key (200, 512, 1024 or 2048).

Note

If the compatibility mode is used (*subopcode2*=KC_NO_SUBOPCODE), the „old“ data structure *kc_encrypt_str* must be used:

struct kc_encrypt_str
`char buf_lth[4];`
`char en_buffer[256];`
`char filler[4];`

11.2.7 KC_GET_OBJECT - Query information

KC_GET_OBJECT allows you to query information on all objects in the configuration and to query the application parameters.

Different kinds of information can be queried. You can control the type of information UTM returns using the *subopcode1* parameter.

The following information is returned by UTM:

- A list of the names of objects of an object type (*subopcode1*=KC_NAME or KC_NAME_NEXT).
- Properties, status and statistical information on the objects of an object type (*subopcode1*=KC_ATTRIBUTES or KC_ATTRIBUTES_NEXT).

  Properties are understood here to mean the parameters that have been set during the configuration of the objects. UTM returns the current values of these parameters, so any modifications by means of administration functions will be reflected in the data returned.

  Status information describes the current status of an object, e.g. whether a connection is currently being set up or a user is currently signed on.

  Statistical information includes counter values and internally measured wait times. UTM returns the following values, for example: the number of messages that the application has exchanged with a partner application of a client since its start, the number of messages being stored temporarily in a partner-specific message queue or the number of program unit runs that have been started using a transaction code.

  The properties of an object and status and statistical information on an object are returned by UTM in the data area in the data structure for the object type (see page 445f). If UTM returns information on several objects, then UTM stores an array of data structures for the object type in the data area.

  Where the properties of an object are discussed in the following text, this refers to object properties, status and statistical information.
- The current settings for the application parameters (*subopcode1*= KC_APPLICATION_PAR)

  The values returned by UTM are dependent on the parameter type you have specified in *obj_type*. You can, for example, choose between the maximum values of the application set during the KDCDEF generation, the system parameters, the current timer settings or statistical information on the current application load. In point 3 on page 286 is a list of the parameter types you may select from.

  For each parameter type there is a data structure in which UTM returns the application parameters queried. The data structures are described on page 591f.

Controlling the output of object names and object properties

UTM returns the object names sorted alphabetically. Accordingly, the properties of the objects are also returned in order of the object names. In *subopcode2* you can specify if UTM is to return the names in ascending (KC_ASCENDING) or descending (KC_DESCENDING) alphabetical order.

Because the amount of information returned from a query of all objects of an object type can be very large, you should limit the amount of information requested. You have the following options available to limit the amount of information:

- You can specify the point in the alphabetical list at which output is to start in the identification area. You can enter any string for this purpose.

  If the string does not correspond to any object name of the object type specified, then UTM starts the output at the next object in the list, meaning the next highest or next lowest object alphabetically, depending on what you specified in *subopcode2*.

  If the string in the identification area corresponds to an object name, then the starting point of the output is dependent on *subopcode1*:
  – for *subopcode1*=KC_NAME and KC_ATTRIBUTES, the output begins with this object.
  – for *subopcode1*=KC_NAME_NEXT and KC_ATTRIBUTES_NEXT, the output begins with the next object, meaning the next highest or next lowest object alphabetically, depending on what you specified in *subopcode2*.

  The list of names or properties output will extend at most to the last (for *subopcode2*=KC_ASCENDING) or to the first (for *subopcode2*= KC_DESCENDING) object in the alphabetically ordered list of objects.

  If the names or properties of the objects are to be read starting with the first object alphabetically of an object type, then you must specify *subopcode2*=KC_ASCENDING and set the identification area to binary zero.

  If the names or properties of the objects are to be read in alphabetically descending order starting with the last object of an object type, then you must specify *subopcode2*=KC_DESCENDING and pass the string X´FF...‘ in the identification area.

- In the *obj_number* field of the parameter area you can specify the maximum number of objects for which UTM is to return information.

- In the selection area you can pass selection criteria to UTM.

  UTM will then only return information on those objects meeting the specified selection criteria. A selection criterion is an object property. You could then, for example, output all the names of clients/printers that are currently connected to the application (*obj_type* =KC_PTERM). A list of all the selection criteria that you can specify can be found under point 9 on page 292.

Using selection criteria, you can target specific objects for selection and can therefore limit the amount of data returned.
The use of selection criteria does, however, influence the performance of the call, especially if only object names are queried. UTM must then read and check the properties for each object to see if each property satisfies its selection criterion. This means that, in this case, a call using selection criteria results in much more work than a call without selection criteria.

The following should be observed when querying information

When querying object names or object properties, information is also returned for objects that have been marked as deleted. You can limit the output to those objects not deleted using the selection criterion *delete*=´N´. With the selection criterion *delete*=´Y´, you can also output all objects of the object type that have been deleted.

Note in the case of UTM cluster applications

- In UTM cluster applications, information is only supplied concerning the objects of the node application at which the call is executed.
- The specifications KC_NO_READ_GSSBFILE and KC_NO_READ_USERFILE in *subopcode2* allow you to determine whether or not the cluster GSSB file or cluster user file are accessed on follow-up calls for objects of type GSSB or USER. This makes it possible to improve performance when there are a large number of follow-up calls.

  If *subopcode2*=KC_NO_READ_GSSBFILE or *subopcode2*=KC_NO_READ_USERFILE then the objects are always supplied in ascending order.

  This improved performance is coupled with a level of uncertainty regarding the information that is returned by the follow-up calls. Since the data is not read again from the file, it may not be up-to-date.

Possible applications

You should consider the following points when using the subopcodes KC_... and KC_..._NEXT:

- You should use KC_ATTRIBUTES or KC_NAME if you want to check whether or not an object with the object name specified already exists. To do this, specify the object name you want in the identification area and enter *obj_number*=1. The return code, with which you can determine whether an object exists (sub-return code = KC_SC_SAME) or not (sub-return code = KC_SC_NEXT), is evaluated after the call.
- You can use KC_ATTRIBUTES or KC_NAME as the "starting point" of a succession of queries if you want to query the object names starting with a certain string but do not know if an object exists for this string.

  For example, the string ´S␣␣␣␣␣␣␣´ can be specified as the name if the objects are to be read starting with the first object name that begins with an "S" (as long as spaces are considered to be the first letter alphabetically).
- In a follow-up call in which you have specified in the identification area that the last object read in the previous call is to be the new starting point (successive query), then KC_ATTRIBUTES and KC_NAME are not suitable for use. For these parameter values the object name specified will be returned. If *obj_number*=1 was specified and you are executing a successive query, then this same object will always be read.
  In this case, you must specify KC_ATTRIBUTES_NEXT or KC_NAME_NEXT. The following object will then be read as the first object.

You will find an example of a successive query of objects on page 304.

KDCINF (page 691)
Less information than with the program interface is returned with KDCINF, however.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identification area**	**selection area**	**data area**
Output the names of all objects of a certain object type	*subopcode1*: KC_NAME_NEXT or KC_NAME *subopcode2*: output in alphabetically ascending or descending order *obj_type*: object type *obj_number*: maximum number of object names	Name of the object with/after which the output of names is to begin	——	A pointer to a data area for the data returned by UTM must be passed in the call.
Output the names of all objects of a certain type with certain properties			Selection criteria used by UTM to limit the amount of data output	
Output properties and statistical information of objects of a certain type with certain properties	*subopcode1*: KC_ATTRIBUTES_NEXT or KC_ATTRIBUTES *subopcode2*: output in alphabetically ascending or descending order *obj_type*: object type *obj_number*: maximum number of objects for which UTM is to output properties.			
Output properties and statistical information of objects of a certain type	*subopcode1*: KC_ATTRIBUTES_NEXT or KC_ATTRIBUTES *subopcode2*: output in alphabetically ascending or descending order *obj_type*: object type *obj_number*: maximum number of objects for which the UTM properties and statistical information are to be output.		——	
Output application parameters	*subopcode1*: KC_APPLICATION_PAR *obj_type*: parameter type *obj_number*: 0	——	——	

[1] The operation code KC_GET_OBJECT must be specified in the parameter area in all cases.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_GET_OBJECT
1.	subopcode1	KC_NAME_NEXT / KC_NAME / KC_ATTRIBUTES_NEXT / KC_ATTRIBUTES / KC_APPLICATION_PAR
2.	subopcode2	KC_ASCENDING / KC_DESCENDING / KC_READ_NO_GSSBFILE / KC_READ_NO_USERFILE / binary zero
3.	obj_type	Object type / parameter type
4.	obj_number	Number of objects / 0
5.	id_lth	Length of the object name in the identification area/ 0
6.	select_lth	Length of the data in the selection area/ 0
7.	data_lth	Length of the data area

Identification area (starting on page 290)

8.	Object name/ —

Selection area (starting on page 292)

9.	Data structure of the object type with selection criteria / —

Data area

—

KDCADMI call

KDCADMI (&parameter_area, &identification_area, &selection_area, &data_area) or
KDCADMI (&parameter_area, &identification_area, NULL, &data_area) or
KDCADMI (&parameter_area, NULL, NULL, &data_area)

Data returned by UTM

Parameter area (starting on page 298)

	Field name	Contents
10.	retcode	Return code
11.	number_ret	Number of objects
12.	data_lth_ret	Length of the return data

Data area (starting on page 301)

13.	Data structures of the object or parameter type / array of object names

1. In *subopcode1* you specify the type of information to be returned by UTM. You can specify the following values:

   KC_NAME

   UTM is to return the names of objects of the object type *obj_type*.

   If the string specified in the identification area matches an object name, then the output is to begin with the name of this object.

   If the string in the identification area does not match an object name of the object type specified, then UTM is to begin the output with the next object, i.e. with the next highest object alphabetically for *subopcode2*=KC_ASCENDING or the next lowest object alphabetically for *subopcode2*= KC_DESCENDING.

   KC_NAME_NEXT

   UTM is to return the names of objects of the object type *obj_type*.

   The output is to begin with the object name following the string specified in the identification area, i.e. with the next highest object alphabetically for *subopcode2*=KC_ASCENDING or the next lowest object alphabetically for *subopcode2*= KC_DESCENDING (see also point 2).

KC_ATTRIBUTES
: UTM is to return properties of objects of the object type *obj_type*.

  If the string specified in the identification area matches an object name, then the output is to begin with the properties of this object.

  If the string in the identification area does not match an object name of the object type specified, then UTM is to begin the output with the next object, i.e. with the next highest object alphabetically for *subopcode2*=KC_ASCENDING or the next lowest object alphabetically for *subopcode2*= KC_DESCENDING.

KC_ATTRIBUTES_NEXT
: UTM is to return properties of objects of the object type *obj_type*.

  The output is to begin with the object whose name follows the name specified in the string, i.e. with the next highest object alphabetically for *subopcode2*=KC_ASCENDING or the next lowest object alphabetically for *subopcode2*= KC_DESCENDING.

KC_APPLICATION_PAR
: UTM is to return the application parameters of the parameter type specified in *obj_type*.

2. The data you must specify in the *subopcode2* field depends on the value specified in *subopcode1*.

   – For *subopcode1*=KC_APPLICATION_PAR you must set *subopcode2* to binary zero.

   – For KC_NAME_NEXT, KC_NAME, KC_ATTRIBUTES_NEXT, and KC_ATTRIBUTES, you must specify one of the two following values in *subopcode2*:

     KC_ASCENDING,
     : UTM returns the information on the objects in alphabetically ascending order according to object name, i.e. the next highest name alphabetically.

     KC_DESCENDING
     : UTM returns the information on the objects in alphabetically descending order according to object name, i.e. the next lowest name alphabetically.

     KC_READ_NO_GSSBFILE
     : This value may only be specified in the case of follow-up calls in a UTM cluster application with object type=KC_GSSB.
       If KC_READ_NO_GSSBFILE is specified, then UTM does not access the cluster GSSB file again but instead uses the data from the last call with KC_ASCENDING. This improves performance when reading GSSBs, see note below.
       UTM returns the information on the GSSBs in ascending object name order.

KC_READ_NO_USERFILE

This value may only be specified in the case of follow-up calls in a UTM cluster application with object type=KC_USER.
If KC_READ_NO_USERFILE is specified, then UTM does not access the cluster user file again but instead uses the data from the last call with KC_ASCENDING. This improves performance when reading large numbers of user IDs, see note.

UTM returns the information on the user IDs in ascending object name order.

If, in UTM cluster applications, you read in GSSBs or user IDs with *subopcode2*=KC_ASCENDING or *subopcode2*=KC_DESCENDING then all the objects are read in locally from the cluster GSSB file or cluster user file and sorted. Each time you reread the GSSBs/user IDs with this *subopcode2*, all the GSSBs (max. 30000) or all the user IDs are again read in and sorted.

If you require a high performance level, only specify KC_ASCENDING for the first call and use KC_READ_NO_GSSBFILE or KC_READ_NO_USERFILE for all follow-up calls. However, this means that any changes made after the first call are not displayed.

3. in the *obj_type* field you must specify the type of the objects or application parameters for which UTM is to return information. The data you must specify in *obj_type* depends on the value specified in *subopcode1*. Please consult the following table for the values

allowed. The meanings of the object/parameter types are described on page 166f.

	Object/parameter type	Permissible specifications for *subopcode1*=
B	*Object type:* KC_ABSTRACT_SYNTAX KC_ACCESS_POINT KC_APPLICATION_CONTEXT KC_BCAMAPPL KC_CON KC_EDIT KC_GSSB KC_KSET KC_LOAD_MODULE KC_LPAP KC_LSES KC_LTAC KC_LTERM KC_MESSAGE_MODULE KC_MUX KC_OSI_ASSOCIATION KC_OSI_CON KC_OSI_LPAP KC_PROGRAM KC_PTERM KC_QUEUE KC_TAC KC_TPOOL KC_TRANSFER_SYNTAX KC_USER KC_USER_DYN1 KC_USER_DYN2 KC_USER_FIX	KC_ATTRIBUTES, KC_ATTRIBUTES_NEXT, KC_NAME, KC_NAME_NEXT
X/W	*Object type:* KC_DB_INFO KC_PTC KC_SFUNC KC_SUBNET KC_TACCLASS	KC_ATTRIBUTES, KC_ATTRIBUTES_NEXT
	Object type: KC_CLUSTER_NODE	KC_ATTRIBUTES

Object/parameter type	Permissible specifications for *subopcode1*=
Parameter type: KC_CLUSTER_CURR_PAR KC_CLUSTER_PAR KC_CURR_PAR KC_DIAG_AND_ACCOUNT_PAR KC_DYN_PAR KC_MAX_PAR KC_MSG_DEST_PAR KC_PAGEPOOL KC_QUEUE_PAR KC_SIGNON KC_SYSTEM_PAR KC_TASKS_PAR KC_TIMER_PAR KC_UTMD_PAR	KC_APPLICATION_PAR

In the case of *obj_type*=KC_USER, KC_USER_DYN1, KC_USER_DYN2 and KC_USER_FIX, please note the following:

The data structures *kc_user_str*, *kc_user_fix_str*, *kc_user_dyn1_str* and *kc_user_dyn2_str* are defined for the object types KC_USER, KC_USER_DYN1, KC_USER_DYN2 and KC_USER_FIX. The fields present in the three data structures *kc_user_fix_str*, *kc_user_dyn1_str* and *kc_user_dyn2_str* are also present in the data structure *kc_user_str*. This subdivision into three data structures was undertaken in order to make it possible to access specific user information values and consequently improve performance, in particular when reading user information in UTM cluster applications.

All the data relating to the cluster user file is located in the data structure *kc_user_dyn2_str*.
To read this data, openUTM must access the cluster user file. That is why, when reading user information in UTM cluster applications, you should preferably use the new object types and only call KC_USER_DYN2 if you currently need the data that this call returns.

The data structure *kc_user_str* for user data and the associated object type KC_USER which represented the only mechanism available in versions < openUTM V6.0 continue to be fully supported. New user properties (as of openUTM V6.0) have also been included in this overall structure, with the result that no modifications are required to user program units which read data via the ADMI interface.

Note the following for *obj_type*=OSI_ASSOCIATION:

– For *subopcode1*=KC_NAME and KC_NAME_NEXT, UTM returns the names of the OSI TP associations set during KDCDEF generation. The names consist of an association prefix specified in an OSI-LPAP command and a serial number.
You can specify an association name in the identification area for these values of *subopcode1*.

– For *subopcode1*=KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT, UTM only returns the properties of associations that belong to a particular partner application and that have been or are currently being established. For this reason, you **must** specify the partner application as a selection criterion when calling the OSI-LPAP partner. You pass the data structure *kc_osi_association_str* containing the name of the OSI-LPAP partner in the selection area (see page 509).

The properties of an association are not stored internally under the association name, but under an association ID assigned by UTM to an association as long as it is in existence. It is not possible to assign an association ID to the name of an association. UTM therefore interprets the string specified in the identification area (field *kc_name8* in the union *kc_id_area*) as an association ID. UTM returns the properties of the active associations to a partner application sorted according to the association IDs. It is not possible to query the properties of an association name.

X/W In the case of *obj_type*=KC_SUBNET, please note the following:

X/W – *subopcode2* must contain binary zero (KC_NO_SUBOPCODE).

X/W – The identification area can be used.

X/W – The selection area may not be specified.

X/W – The output of the information on the subnets is not sorted according to the subnet
X/W names (*mapped_name*), but takes place in the order in which the statements were
X/W specified during generation - separated according to IPv4 and IPv6 subnets.

4. In *obj_number* you can specify the number of objects for which UTM is to return information. The following can be specified:

– For *subopcode1*=KC_NAME,KC_NAME_NEXT, KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT:
obj_number specifies the maximum number of objects for which UTM is to return information.
If you specify *obj_number*=0, then UTM will return information on as many objects as will fit in the data area, or less if there are no more objects of the object type available.
In the case of *obj_type*=KC_CLUSTER_NODE, please note the following:
If you specify an *obj_number* > 32, openUTM sets *obj_number* to 32.

– For *subopcode1*=KC_APPLICATION_PAR you must always specify *obj_number*=0.

5. The data you must specify in the *id_lth* is dependent on the data contained in the *subopcode1* field:

   – For *subopcode1*=KC_NAME, KC_NAME_NEXT, KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT:

     In *id_lth* you must specify the length of the data structure you have passed to UTM in the identification area.

   – For *subopcode1*=KC_APPLICATION_PAR you must always set *id_lth*=0. The contents of the identification area are irrelevant.

6. In *select_lth* you must specify a value ≠0 if you want to pass selection criteria to UTM in the selection area.

   For *subopcode1*=KC_APPLICATION_PAR you may not pass any selection criteria to UTM and must therefore always set *select_lth*=0 in this case.

   For *subopcode1*= KC_ATTRIBUTES or KC_ATTRIBUTES_NEXT and *obj_type*= KC_OSI_ASSOCIATION, you **must** pass the data structure *kc_osi_association_str* with the name of an OSI-LPAP partner in the selection area. In this case, the length of the data structure *kc_osi_association_str* is to be specified in *select_lth*.

X/W For *obj_type*=KC_SUBNET you must always specify *select_lth*=0.

7. In *data_lth* you must specify the length of the data area that you are providing to UTM for returning the information queried.

   – For *subopcode1*= KC_NAME, KC_NAME_NEXT, KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT, the following is true:

     If you specify *obj_number*≠0, then the data area provided for returning the number of objects requested must be large enough. For *obj_number*=n (see 4.) you must specify in *data_lth* a minimum length of (n∗ maximum length of the object name) or (n ∗ length of the data structure of the object type in *obj_type*).

   – For *subopcode1*=KC_APPLICATION_PAR, you must specify at least the length of the data structure of the parameter type set in *obj_type*.

8. The data you must specify in the identification area is dependent on the data contained in the *subopcode1* field and the value of *obj_type*:

   – For *subopcode1*=KC_NAME,KC_NAME_NEXT, KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT:

     You must pass a string to UTM in the identification area. The string specifies the object at which UTM is to begin outputting information.

You can also pass binary zero or a string containing non-printable characters in the identification area. UTM takes the string as it is and searches for the next highest (for *subopcode2*=KC_ASCENDING) or next lowest (for *subopcode2*= KC_DESCENDING) object name.

You place a *kc_id_area* union (see page 171) in the identification area. The string must be passed in the union element that belongs to the object type specified in *obj_type*.

– For *obj_type*=KC_PROGRAM and KC_LOAD_MODULE:
you pass the string in the element *kc_name32*. The name must be left-justified, and the rest of the field must either be padded with blanks or end with the null byte (\0).
The string specified does not have to be an object name.

– For *obj_type*=KC_CON, KC_PTERM and KC_MUX:
you must pass the string in the union element *kc_triple_str*. A name triplet (object name, computer name, name of the local application) can be specified in *kc_triple_str*. Each of the names can be up to 8 characters long.
You can specify any string for each of the three names. The name does not need to exist. It is sufficient just to specify a string for the object name, you do not need to specify the computer name and the name of the local application. You may set these to binary zero.
When evaluating the strings in the identification area, UTM interprets the three names as a 24 character long object name. The starting point of the output is determined accordingly.

– For *obj_type*=KC_SFUNC
you can do without the specifications in the identification area, or you specify a valid function key in the union element *kc_name4*. If you use a different string, the call will be rejected.
The following options are valid:
B X/W on BS2000 systems: F1 to F20 and K1 to K14
on Unix, Linux and Windows systems: F1 to F20
If you do not make an entry in the identification area, UTM will return data on all function keys.
If you enter a valid function key, UTM will start output with that function key

– For *obj_type* = KC_TACCLASS:
you can specify the values of an existing TAC class, a LOW VALUE or a HIGH VALUE in the union element *kc_name2*. If you specify any other string, the call will be rejected.

– For *obj_type* = KC_OSI_ASSOCIATION
you must pass the string in the union element *kc_name8*.
For *subopcode1*=KC_NAME and KC_NAME_NEXT,
UTM interprets the string as the name of an OSI TP association.

For *subopcode1*= KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT,
UTM interprets the string as an association ID. See the description in point 3 on page 286.

– For *obj_type*=KC_CLUSTER_NODE
you must pass LOW VALUE, HIGH VALUE or empty fields in the identification area. Otherwise the call is rejected. No specific node is addressed. Choose a value for *data_lth* that is large enough for information to be passed to all the nodes.
For all other object types, the string must be passed in the union element *kc_name8*. The string must be stored left-justified and the rest of the field is to be padded with blanks.
The string specified does not have to be an object name.

X/W – If the identification area is used for *obj_type*=KC_SUBNET, the name specified there must be an object name, i.e. it must correspond to a generated subnet name (*mapped_name*) as the information on the subnets is not sorted in alphabetical order when it is stored. If no generated *mapped_name* is specified, KC_MC_NO_ELT is returned as the return code with subcode KC_SC_NOT_EXISTENT.

– For *subopcode1*=KC_APPLICATION_PAR the null pointer should be passed for the identification area.

9. In the selection area, if *select_lth* ≠0, then you must pass the data structure of the object type to UTM together with the selection criteria. The rest of the fields in the data structure must be set to binary zero.
The data structures are described in section “Data structures for describing object properties” on page 445f. The name of each data structure is created as follows: data structure *"typ_str*" belongs to the object type "*TYP*", so, for example, the data structure *kc_lterm_str* belongs to KC_LTERM.

   If *select_lth* = 0, the selection area, and therefore the selection criteria, are not evaluated.

   A selection criterion is an object property. If selection criteria are specified, then UTM executes a selective search of the objects. Only information on the objects meeting the selection criteria is returned. The selection criteria you may specify for each object type is listed in the following text.

Possible selection criteria

- *obj_type*=KC_CON: connections to LU6.1 partner applications

  In the selection area you pass the data structure *kc_con_str* with the selection criteria. The following data may be specified:

Field name	Meaning
connect_mode=´Y´	UTM is only to return information on LU6.1 connections currently open.
pronam	UTM is only to return information on LU6.1 connections to partner applications that are running on a certain computer. You specify the name of the computer in *pronam*.

  You can also specify both selection criteria together, meaning you can specify both fields at the same time.

- *obj_type*=KC_LPAP: LPAP partner

  In the selection area, you pass the data structure *kc_lpap_str* with the selection criteria. The following specifications are permitted:

Field name	Meaning
master	*master* contains the name of a master LPAP in an LPAP bundle. UTM returns information on the slave LPAPs in this LPAP bundle.

- *obj_type*=KC_LSES: sessions to LU6.1 partner applications

  In the selection area you pass the data structure *kc_lses_str* with the selection criteria. The following data may be specified:

Field name	Meaning
connect_mode=´Y´	UTM is only to return information on sessions for which a transport connection is currently established.
lpap	UTM is only to return information on sessions that are assigned to a certain LU6.1 partner application. You specify the name of the LPAP partner assigned to this partner application in *lpap*.
delete	*delete*=´Y´: UTM is only to return information on sessions that were deleted from the configuration. *delete*=´N´: UTM is only to return information on sessions that were not deleted from the configuration.

  You can also specify both selection criteria, meaning you can specify both fields at the same time.

- *obj_type*=KC_LTERM: LTERM partner

  In the selection area, you pass the data structure *kc_lterm_str* with the selection criteria. The following specifications are permitted:

Field name	Meaning
master	*master* contains the name of a master LTERM in an LTERM bundle: UTM returns information on the slave LTERMs of the LTERM bundle for the specified master LTERM. *master* contains the name of a primary LTERM in an LTERM group: UTM returns information on the group LTERMs of the LTERM group for the specified primary LTERM.

- *obj_type*=KC_MUX: multiplex connections

  In the selection area you pass the data structure *kc_mux_str* with the selection criteria. The following data may be specified:

Field name	Meaning
connect_mode=´Y´	UTM is only to return information on multiplex connections for which a transport connection to the message router is currently established.
pronam	UTM is only to return information on multiplex connections that are defined for message routers on a certain computer. You specify the name of the computer in *pronam*.

  You can also specify both selection criteria, meaning you can specify both fields at the same time.

- *obj_type*=KC_OSI_ASSOCIATION: associations to OSI TP partner applications

  For *subopcode1*= KC_NAME and KC_NAME_NEXT, no selection criterion may be specified.

  For *subopcode1*= KC_ATTRIBUTES and KC_ATTRIBUTES_NEXT, you **must** pass the following selection criterion to UTM (see the description under point 3 on page 286). To do this, pass the data structure *kc_osi_association_str* in the selection area with the following data:

Field name	Meaning
osi_lpap	UTM is only to return information on associations that are assigned to a certain OSI TP partner application. You specify the name of the OSI-LPAP partner assigned to this partner application in *osi_lpap*.

- *obj_type*=KC_OSI_LPAP: Properties of OSI TP partner applications

  In the selection area, you pass the data structure *kc_osi_lpap_str* with the selection criteria. The following specifications are permitted:

Field name	Meaning
master	*master* contains the name of a master LPAP in an OSI-LPAP bundle: UTM returns information on the slave LPAPs of the LPAP bundle for the specified master LPAP.

- *obj_type*=KC_PROGRAM: program units

  In the selection area you pass the data structure *kc_program_str* with the selection criteria. The following data may be specified:

Field name	Meaning
load_module	UTM is only to return information on program units and VORGANG exits that are linked into a certain load module / shared object/DLL. You specify the name of the load module / shared object /DLL in *load_module*.
delete	*delete*=´Y´:UTM is only to return information on program units that have been deleted from the configuration. *delete*=´N´: UTM is only to return information on program units that have not been deleted from the configuration.

  You can also specify both selection criteria, meaning you can specify both fields at the same time.

- *obj_type*=KC_PTERM: clients and printers

  In the selection area you pass the data structure *kc_pterm_str* with the selection criteria. The following data may be specified:

Field name	Meaning
lterm	Is only useful for printers: UTM is to return information on the printers in a printer pool. The printers in a printer pool are assigned to the same LTERM partner. The name of the LTERM partner is to be specified in *lterm*.
connect_mode=´Y´	UTM is only to return information on clients/printers that are currently connected to the application.
pronam	UTM is only to return information on clients and printers running on a certain computer or which are connected to this computer. You specify the name of the computer in *pronam*.
delete	*delete*=´Y´:UTM is only to return information on clients and printers that have been deleted from the configuration. *delete*=´N´: UTM is only to return information on clients and printers that have not been deleted from the configuration.

  You may only specify the selection criterion *lterm* alone. All other fields of the data structure must then be set to binary zero.
  Either *connect_mode* and *pronam* or *pronam* and *delete* can be specified together, meaning the fields can be set at the same time. *connect_mode* and *delete* cannot be set at the same time.

- *obj_type*=KC_USER, KC_USER_DYN1, KC_USER_DYN2, KC_USER_FIX: user IDs

  In the selection area you pass the data structure *kc_user_str* or *kc_user_dyn1_str* with the selection criteria. The following data may be specified:

Field name	Meaning
connect_mode=´Y´	UTM is only to return information on user IDs with which a user/client is currently signed on to the application.
delete	*delete*=´Y´: UTM is only to return information on user IDs that have been deleted from the configuration. *delete*=´N´: UTM is only to return information on user IDs that have not been deleted from the configuration.

  The selection criteria must not be specified together, i.e. only one field may be set per call.

- *obj_type*=KC_LTERM, KC_LTAC or KC_TAC: LTERM partner or transaction codes.

  In the selection area you pass the data structure *kc_ltac_str* (KC_LTERM), *kc_ltac* (KC_LTAC) or *kc_tac_str* (KC_TAC) with the selection criteria. The following data may be specified:

Field name	Meaning
delete	*delete*=´Y´:UTM is only to return information on objects that have been deleted from the configuration. *delete*=´N´: UTM is only to return information on objects that have not been deleted from the configuration.

10. in the *retcode* field UTM outputs the return codes of the call. In addition to the return codes listed in section “Return codes” on page 174, the following return codes can also be returned.

Main code = KC_MC_OK The call was processed without error. **Subcodes:**
KC_SC_SAME *subopcode1* = KC_NAME or KC_ATTRIBUTES was set, and an object exists that corresponds to the object name specified in the identification area. This object is passed in the data area as the first object.
KC_SC_NEXT *subopcode1* = KC_NAME_NEXT or KC_ATTRIBUTES_NEXT was set. Or *subopcode1* = KC_NAME or KC_ATTRIBUTES was set but no object exists that corresponds to the object name specified in the identification area. The next highest or next lowest object (depending on *subopcode2*) is passed in the data area as the first object.

Main code = KC_MC_LAST_ELT The call was processed without error, but fewer objects were read than were queried, and the last object has already been reached. **Subcodes:**
KC_SC_SAME *subopcode1* = KC_NAME or KC_ATTRIBUTES was specified. An object corresponding to the object name specified in the identification area exists. UTM has written object names or properties to the data area, but for fewer objects than were requested in *obj_number* or (for *obj_number* = 0) for fewer objects than could fit in the space provided in the data area passed. The last or first object, respectively, was reached beforehand.
KC_SC_NEXT *subopcode1* = KC_NAME_NEXT or KC_ATTRIBUTES_NEXT was set. Or *subopcode1* = KC_NAME or KC_ATTRIBUTES was set but no object exists that corresponds to the object name specified in the identification area. The next highest or next lowest object (depending on *subopcode2*) is passed in the data area as the first object. UTM has written object names or properties into the data area, but for fewer objects than were requested in *obj_number* or (for *obj_number* = 0) for fewer objects than could fit in the space provided in the data area passed. The last or first object, respectively, was reached beforehand.

Main code = KC_MC_NO_ELT
subopcode1 = KC_NAME, KC_NAME_NEXT, KC_ATTRIBUTES or KC_ATTRIBUTES_NEXT was specified. There is no element or no next element corresponding to the object name specified.
Subcode:

KC_SC_NO_INFO

X/W
X/W
KC_SC_NOT_EXISTENT
The object name specified in the identification area was not found in *obj_type*.

Main code = KC_MC_MEMORY_INSUFF
UTM cannot execute the function because that would require more internal storage space than UTM has available.
Subcode:

KC_SC_NO_INFO

Main code = KC_MC_REJECTED
The call was rejected by UTM because no object of the specified type exists.
Subcode:

KC_SC_NOT_GEN
If *obj_type*=KC_GSSB, then no global secondary storage areas exist at the present time.
If *obj_type* = KC_MESSAGE_MODULE, then the application was generated without the KDCDEF control statement MESSAGE.
If *obj_type* = KC_UTMD_PAR, then the application was generated without the KDCDEF control statement UTMD.
If *obj_type* =KC_TACCLASS, then no TAC class was created during the KDCDEF generation.
If *obj_type*=KC_SUBNET, either no IP subnet was generated (Unix, Linux, or Windows systems) or no subnet generation is possible (BS2000 systems).

KC_SC_NO_F_KEYS_GENERATED
You specified *obj_type*=KC_SFUNC, but no function keys were generated for the application.
(See the openUTM manual “Generating Applications”)

KC_SC_CCFG_FILE_READ_ERROR
Only in UTM cluster applications:
You have specified *obj_type*=KC_CLUSTER_PAR or KC_CLUSTER_NODE in order to obtain information about a UTM cluster application. An error occurred while reading the cluster configuration file.

KC_SC_CCFG_INVAL_NODE_BUFF_LTH
Only in UTM cluster applications:
Internal UTM error.
Please contact system support.

Main code = KC_MC_REJECTED The call was rejected by UTM because no object of the specified type exists. **Subcode:**
KC_SC_CCFG_FILE_LOCK_ERROR Only in UTM cluster applications: The cluster configuration file is locked.
KC_SC_CCFG_RT_CODE_NOT_OK Only in UTM cluster applications: Internal UTM error. Please contact system support.
KC_SC_CUSF_USER_NOT_FOUND Only in UTM cluster applications: Specified user does not exist.
KC_SC_CUSF_RT_CODE_NOT_OK Only in UTM cluster applications: Internal UTM error. Please contact system support.

Main code = KC_MC_NOT_EXISTENT The object specified does not exist. **Subcode:**
KC_SC_NO_INFO *obj_type*=KC_TACCLASS: no valid function key or TAC class was specified in the identification area.

Main code = KC_MC_SEL_INVALID Invalid data was specified in the selection area. **Subcode:**
KC_SC_NO_INFO

11. After a call with *subopcode1*=KC_NAME, KC_NAME_NEXT, KC_ATTRIBUTES or KC_ATTRIBUTES_NEXT, *number_ret* contains the number of objects for which UTM has returned information in the data area.

    If no more objects corresponding to the string specified in the identification area exist, then UTM returns *number_ret*=0 and *data_lth_ret*=0 and sets the corresponding return code.

    After a call with *subopcode1*=KC_APPLICATION_PAR, UTM always returns *number_ret*=0.

12. In *data_lth_ret* UTM returns the length of the data that UTM has stored in the data area.

    The length of the data returned is:

    – for *subopcode1*=KC_NAME, KC_NAME_NEXT:
      number of objects ∗ length of the name field belonging to the object type

    – for *subopcode1*= KC_ATTRIBUTES or KC_ATTRIBUTES_NEXT:
      number of objects ∗ length of the data structure of the object type

    – for *subopcode1*=KC_APPLICATION_PAR:
      length of the data structure of the parameter type

    If no object or no more objects corresponding to the string specified in the identification area exist, then UTM returns *data_lth_ret*=0 and sets the corresponding return code.

13. in the data area UTM returns the information queried.

    – *subopcode1*=KC_NAME, KC_NAME_NEXT:

      UTM returns an array of object names. The object names are ordered alphabetically in the array in ascending (for *subopcode2*=KC_ASCENDING) or descending (for *subopcode2*=KC_DESCENDING) order.

      The length of the individual names corresponds to the length of the name field in the data structure of the object type.

      For *obj_type*=KC_CON, KC_PTERM and KC_MUX, UTM returns an array of name structures with the following format:

struct kc_triple_str
`char p_name[8];`
`char pronam[8];`
`char bcamappl[8];`

      The three fields of the data structure contain the following for each of the objects:

      p_name
      : object name, i.e. the name of the connection, client, printer or multiplex connection

      pronam
      : Name of the computer on which the object is located

      bcamappl
      : Name of the local application via which the connection to this object has been established.

For *subopcode1*=KC_NAME_NEXT the name array always begins with the object name that is the next highest or next lowest alphabetically, depending on the value of *subopcode2*, with respect to the string specified in the identification area.
There are two cases for *subopcode1*=KC_NAME:

If an object name exists that corresponds to the string you have specified in the identification area, then the name array begins with this object name. UTM returns the KC_SC_SAME return subcode.

If the string specified in the identification area does not correspond to an object name, then, just as with *subopcode1*=KC_NAME_NEXT, the name array begins with the object name that is the next highest or next lowest alphabetically, depending on the value of *subopcode2*, with respect to the string specified in the identification area. UTM returns the KC_SC_NEXT return subcode.

– *subopcode1*= KC_ATTRIBUTES or KC_ATTRIBUTES_NEXT:

UTM places an array of data structures of the object type in the data area. Each data structure contains the properties of an object. The data structures are placed one after the other and are put in ascending or descending alphabetical order according to the object names, depending on the value of *subopcode2*.

The data structures are described in section “Data structures for describing object properties” on page 445f. The name of each data structure is created as follows: the data structure *"typ*_str" belongs to the object type "*TYP*", so, for example, the data structure *kc_lterm_str* belongs to KC_LTERM.

In the data structures, the fields that were not specified when the object was added to the configuration contain the default values, blanks or ´0´. Fields only relevant to other operating systems are set to binary zero.

The object with which the array begins depends on the value of *subopcode1* and on the name specified in the identification area.

For *subopcode1*=KC_ATTRIBUTES_NEXT the array begins with the object that is the next highest or next lowest alphabetically, depending on the value of *subopcode2*, with respect to the string specified in the identification area.

There are two cases for *subopcode1*=ATTRIBUTES:
If an object name exists that corresponds to the string you have specified in the identification area, then the name array begins with this object name. UTM returns the KC_SC_SAME return subcode.

If the string specified in the identification area does not correspond to an object name, then, just as with *subopcode1*=KC_ATTRIBUTES_NEXT, the name array begins with the object name that is the next highest or next lowest alphabetically, depending on the value of *subopcode2*, with respect to the string specified in the identification area. UTM returns the KC_SC_NEXT return subcode.

– *subopcode1*= KC_APPLICATION_PAR:

  UTM places the data structure of the parameter type specified in *obj_type* in the data area. UTM returns the application parameters requested in the data structure.

  The data structures are described in section “Data structures used to describe the application parameters” on page 591f. The name of each data structure is created as follows: the data structure *"typ_str"* belongs to the object type "*TYP*", so, for example, the data structure *kc_max_par_str* belongs to KC_UKC_MAX_PAR.

Example of a successive query with KC_ATTRIBUTES_NEXT

Task

All information on user IDs whose names begin with "S" is to be read. It is assumed in the following that such user IDs exist.

Solution

1. KC_GET_OBJECT call:
   (It is assumed that n objects are found by this call, i.e. that *n_ret*=*n*.)

Data to be entered in the parameter area:	
version	KC_ADMI_VERSION_1
retcode	KC_RC_NIL
version_data	KC_VERSION_DATA
opcode	KC_GET_OBJECT
subopcode1	KC_ATTRIBUTES
subopcode2	KC_ASCENDING
obj_type	KC_USER
obj_number	n
id_lth	8
select_lth	0
data_lth	n ∗ sizeof(struct kc_user_str)
Data to be entered in the identification area:	
´S␣␣␣␣␣␣␣´ or ´S\0´ (\0 = null byte in C)	
Data to be entered in the selection area:	
none	
Data to be entered in the data area:	
none	

Data returned by UTM:

Data returned in the parameter area:	
retcode	KC_MC_OK with subcode KC_SC_SAME or KC_SC_NEXT
number_ret	n_ret
data_lth_ret	n_ret∗sizeof(struct kc_user_str)
Data returned in the data area:	
n_ret ∗ data structure *kc_user_str* with the properties of the user IDs	

If the last user ID returned still begins with "S", then another call must be made.

2. KC_GET_OBJECT call:
   (Data to be entered which differs from that in the first call is underlined)

Data to be entered in the parameter area:	
version	KC_ADMI_VERSION_1
retcode	KC_RC_NIL
version_data	KC_VERSION_DATA
opcode	KC_GET_OBJECT
subopcode1	KC_ATTRIBUTES_NEXT
subopcode2	KC_ASCENDING
obj_type	KC_USER
obj_number	n
id_lth	8
select_lth	0
data_lth	n ∗ sizeof(struct kc_user_str)
Data to be entered in the identification area:	
Name of the last user ID returned by UTM in the first call	
Data to be entered in the selection area:	
none	
Data to be entered in the data area:	
none	

Data returned by UTM:

Data returned in the parameter area:	
retcode	KC_MC_OK with subcode KC_SC_NEXT [1]
number_ret	n_ret (≤ n)
data_lth_ret	n_ret ∗ sizeof(struct kc_user_str)
Data returned in the data area:	
n_ret ∗ data structure *kc_user_str* with the data of the user IDs	

[1] The return codes KC_MC_LAST_ELT (if less than n objects were found) and KC_MC_NO_ELT (if no further object was found) can also occur.

The second call is repeated until all user IDs beginning with "S" have been read. Whether or not all user IDs beginning with "S" have been read can be determined by evaluating the return data. This means that if the name of the last user ID returned by UTM begins with "S", then the call must be repeated again. If it does not begin with "S" or if *number_ret* ≠ *obj_number* in the last call, then all user IDs beginning with "S" have been read.

11.2.8 KC_LOCK_MGMT - Release locks in UTM cluster applications

You can use KC_LOCK_MGMT to:

- Sign off all users or an individual user who are/is signed on at an abnormally terminated node application (KDCOFF). Any service data for this user that is valid globally in the cluster is lost when you do this.

  For this function, you use the sub-opcodes KC_SIGNOFF_ALL and KC_SIGNOFF_SINGLE.

- For all users or an individual user who have/has a service bound to a terminated node application, you can mark this service for abnormal termination and this way make it possible for the users or user to sign on again at another node application. The bound service is terminated abnormally the next time the node application to which it is bound is started.

  For this function, you use the sub-opcodes KC_ABORT_BOUND_SERVICE, KC_ABORT_ALL_BOUND_SERVICES and KC_ABORT_PTC_SERVICE.

- Release a cluster user file lock set on an abnormally terminated KDCDEF run (subopcode KC_UNLOCK_USF).

Period of validity / transaction management /clusters

The call permanently modifies the cluster user file. The modification takes effect immediately and cannot be undone by rolling back the transaction.

This function is only available for UTM cluster applications.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_LOCK_MGMT
1.	subopcode1	KC_ABORT_ALL_BOUND_SERVICES / KC_ABORT_BOUND_SERVICE / KC_ABORT_PTC_SERVICE / KC_SIGNOFF_ALL / KC_SIGNOFF_SINGLE / KC_UNLOCK_USF
	id_lth	0
	select_lth	0
2.	data_lth	Length of the data structure / 0

Identification area

—

Selection area

—

Data area

3.	Data structure / 0

KDCADMI-Aufruf

```
KDCADMI (&parameter_area, NULL, NULL, &data_area)
```

Data returned by UTM

Parameter area

	Field name	Content
4.	retcode	Return codes

1. In *subopcode1*, you specify the action that openUTM is to perform. You can specify the following subcodes:

   KC_ABORT_ALL_BOUND_SERVICES
   : Marks all the services that are bound to a terminated node application for abnormal termination. This allows the corresponding users to sign on at other node applications (KDCSIGN). The bound services are terminated abnormally the next time the node application to which they are bound is started.

   KC_ABORT_BOUND_SERVICE
   : Marks a user service that is bound to a terminated node application for abnormal termination. This allows the user to sign on at another node application (KDCSIGN). The bound service is terminated abnormally the next time the node application to which it is bound is started.

   KC_ABORT_PTC_SERVICE
   : Marks a user service that is bound to a terminated node application and has a transaction in PTC state for abnormal termination. This allows the user to sign on at another node application (KDCSIGN). The bound service is terminated abnormally the next time the node application to which it is bound is started.

   KC_SIGNOFF_ALL
   : Sign off all users who are signed on at an abnormally terminated node application so that these users can sign on at another node application. Service data that is valid throughout the cluster for these users is lost.

   KC_SIGNOFF_SINGLE
   : Sign off a single user who is signed on at an abnormally terminated node application so that this user can sign on at another node application. Service data that is valid throughout the cluster for this user is lost

   KC_UNLOCK_USF
   : Releases the lock in the cluster user file after a KDCDEF run was terminated abnormally. It is only necessary to issue the call with subopcode KC_UNLOCK_USF if a KDCDEF has terminated abnormally and a subsequent KDCDEF run outputs message K516 with error code 8.

2. In *data_lth*, enter the length of the data structure in the data area or 0.

3. In the data area, you must specify the data structure *kc_lock_mgmt_str* for all *subopcode1* values excluding KC_UNLOCK_USF:

   The data structure *kc_lock_mgmt_str* is defined as follows:

struct kc_lock_mgmt_str
`char mg_name[8];` `char mg_node[4];`

   The fields in the data structure have the following meanings:

   mg_name –
   Only for subopcode1=KC_SIGNOFF_SINGLE:
   Name of the user who is to be signed off.
   – If *subopcode1*=KC_ABORT_BOUND_SERVICE:
   Name of the user with service which is bound to a terminated node application and is to be marked for abnormal termination.
   – If *subopcode1*=KC_ABORT_PTC_SERVICE:
   Name of the user with a service in the PTC state which is bound to a terminated node application and is to be marked for abnormal termination.
   – Other values for *subopcode1*: irrelevant
   You do not need to specify the node number. openUTM identifies this.

   mg_node
   – Only for subopcode1=KC_SIGNOFF_ALL:
   Number of the node from which all the users are to be signed off.
   – If *subopcode1*=KC_ABORT_ALL_BOUND_SERVICES:
   – Number of the node that was terminated. All the service bound to this node should be marked for abnormal termination.
   – Other values for *subopcode1*: irrelevant

4. openUTM indicates the return code from the call in the *retcode* field. Alongside the return codes listed in section “Return codes” on page 174, the following return codes may also occur:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
B B B KC_SC_CUSF_TRANSIENT_ERROR For each *subop*code1: Temporary error when accessing the cluster user file; please repeat the call.
KC_SC_CUSF_RT_CODE_NOT_OK For each *subopcode1*: Internal UTM error. Please contact system support.
KC_SC_CUSF_INVALID_STATE For *subopcode1*= KC_SIGNOFF_ALL/KC_ABORT_ALL_BOUND_SERVICES: The specified node application has never been started or is currently running. The call can only be executed in the node statuses FAIL or ABTERM. For *subopcode1*= KC_SIGNOFF_SINGLE: The node application at which the specified user is signed in is currently running. If *subopcode1*= KC_ABORT_BOUND_SERVICE/KC_ABORT_PTC_SERVICE: The node application to which the service of the specified user is bound is currently running.
KC_SC_CUSF_USER_HAS_NO_BND_SRV For *subopcode1*=KC_ABORT_BOUND_SERVICE: The user has no bound service.
KC_SC_CUSF_USER_HAS_NO_PTC For *subopcode1*=KC_ABORT_PTC_SERVICE: The user has no node-bound service with a transaction in the PTC state.
KC_SC_CUSF_USER_HAS_PTC For *subopcode1*=KC_ABORT_BOUND_SERVICE: The user has a node-bound service with a transaction in the PTC state.
KC_SC_CUSF_USER_NOT_FOUND For *subopcode1*=KC_SIGNOFF_SINGLE/KC_ABORT_BOUND_SERVICE/ KC_ABORT_PTC_SERVICE: The user was not found.
KC_SC_CUSF_USER_NOT_SIGNED For *subopcode1*=KC_SIGNOFF_SINGLE: The user is not signed in at any node.
KC_SC_DATA_MISSING *mg_name* is not binary zero and *subopcode1*=KC_SIGNOFF_ALL, KC_ABORT_BOUND_SERVICE or KC_ABORT_ALL_BOUND_SERVICES.
KC_SC_NOT_NULL *mg_node* is not binary zero and *subopcode1*=KC_SIGNOFF_SINGLE, KC_ABORT_BOUND_SERVICE or KC_ABORT_PTC_SERVICE.

11.2.9 KC_MODIFY_OBJECT - Modify object properties and application parameters

KC_MODIFY_OBJECT allows you to modify application parameters and object properties and perform other operations on application objects. You can make the following modifications:

Actions for the application's objects

- establish or shut down connections to clients, printers, partner applications
- initiate automatic connections to clients, printers, partner applications
- disable and enable clients, printers, partner applications, user IDs, including their queues, transaction codes and TAC queues
- modify the assignment between client/printer and LTERM partner
- modify the password for a user ID
- change keys in key sets
- alter the timer for monitoring idle states during a session, or deactivate monitoring
- activate or deactivate the BCAM trace for specific objects and users
- replace an application program's load modules or shared objects / DLLs
- Exchange the master LTERMs of two LTERM bundles or add the LTERM to an LTERM group
- Specify that queued messages are to be stored in the dead letter queue (TAC queue KDCDLETQ)

BB
- mark load modules which are loaded in common memory pools for replacement with KC_CHANGE_APPLICATION

BB
- modify the maximum number of clients that can be connected concurrently to the application through a multiplex connection
- modify the computer name and filebase name of a node application

Actions for the application parameters

- change the application timers
- reset the statistics data
- modify maximum values for the application
- activate and deactivate diagnostic functions (e.g. BCAM trace)
- define the number of processes (TASKS) that are to run for the application
- set the maximum number of processes that asynchronous jobs or services with blocking function calls (e.g. KDCS call PGWT) can process concurrently.
- modify the timers for the reciprocal monitoring of the node applications
- in UTM cluster applications, reset the statistics values for the utilization of the cluster page pool

Passing new object properties and application parameter values

Data structures for passing new object properties or application parameters are available in the header file *kcadminc.h*. Each object type and each parameter type has its own data structure. The name of the data structure matches that of the object type/parameter type (in lowercase) with the suffix "_str" (*objecttyp_str*, *parametertyp_str*). The following description specifies the fields to which you must pass the new properties. You will find a complete description of the data structures in section "Data structures used to pass information" on page 443.

The following points should be noted when modifying object properties or application program parameters

When modifying object properties, you can only modify the properties of one object with one KC_MODIFY_OBJECT call.
You must specify the full object name in the identification area so that UTM can unambiguously identify the object.
Object names cannot be modified.

When modifying application parameters, you can modify all parameters belonging to the same parameter type, i.e. which are contained in a single data structure, within a single call.

The transactional modifications specified in a KC_MODIFY_OBJECT call are either made in their entirety or not at all. This does not apply for changes which are not subject to transaction management.

Period of validity / transaction management / cluster

The time at which a modification takes effect and the period for which it is applicable depend on the type of modification. The type of modification also determines whether or not it is subject to transaction management.

The following applies in a UTM cluster application:
The call can initiate actions which either have an effect either globally in the cluster or locally in the node. Actions with a global effect apply to all the node applications in the UTM cluster application irrespective of whether they are currently active or not. Actions with a local effect only apply to the node applications at which they are performed. Depending on the object, all its parameters apply either globally or locally or have a mixed global/local effect. The change may continue to apply beyond the current application run or may apply only to the current run. Modifications which have an impact on the UTM configuration always apply globally to the cluster to ensure that the generation remains consistent. Global validity is indicated by a "G" in the KC_MODIFY_OBJECT operation code column. If no "G" is present in the ID then the effect in a UTM cluster application is local to the node.
A detailed description of the scope of validity of the individual parameters of each object can be found in the description of the data structures.

The following types of modification may occur:

IR/GIR
: The modification is not subject to transaction management. It takes effect immediately (**I**mmediate), and applies only to the current application/UTM cluster application run (**R**un). A RSET call issued in the same transaction but after the modification rolls back the modification.

ID/GID
: The modification is not subject to transaction management. It takes effect immediately (**I**mmediate) and, regardless of the generation version (UTM-S or UTM-F), it applies beyond the current application/UTM cluster application run (**D**urable). A RSET call issued in the same transaction but after the modification rolls back the modification.

PR/GPR
: The modification is subject to transaction management. It takes effect after the end of transaction (**P**END) and it applies only to the current application/UTM cluster application run (**R**un). It can be rolled back with a RSET call issued in the same transaction.

P/GP
: The modification is subject to transaction management. It takes effect after the end of transaction (**P**END) and its duration depends on the generation version of the application. In the case of UTM-F, it only applies to the current application run, with UTM-S, however, it goes beyond the current application run. It can be rolled back within the same transaction with a RSET call.

PD/GPD
: The modification is subject to transaction management. It takes effect after the end of transaction (**P**END) and, independent of the generation version, its effect goes beyond the current application/UTM cluster application run (**D**urable). It can be rolled back within the same transaction with a RSET call.

A/GA
: This generates an announcement (**A**nnouncement), which causes the desired modification (e.g. establishment of a connection/disconnection or replacement of application program) . When the job is executed depends on the load on the application. You can only tell whether the job was executed successfully or not in an information query issued later (e.g. using KC_GET_OBJECT). The job cannot be rolled back.

Note on period of validity in UTM cluster applications:

- If the modification cannot be generated then the administrative modification continues to apply even when a node application is started with a new generation, but persists no later than the end of the UTM cluster application run. The UTM cluster application run begins with the start of the first node application and terminates with the end of the last node application.
- If the modification can be generated, then the generation value and not the administratively modified value applies when a node application is started with a new generation.

The description of the possible modifications under point 6 on page 319ff tells you to which modification type the various modifications belong. The abbreviations listed above are used.

You can also perform some of the modifications using the administration commands. The description under point 6 on page 319ff identifies the commands concerned.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area[1]	**identification area**	**selection area**	**data area**
Modification of object properties	*obj_type*: object type	Name of object	——	Data structure of the object type with the new values of the properties
Modification of application parameters	*obj_type*: parameter type	——	——	Data structure of the parameter type with the new parameter values

[1] The operation code KC_MODIFY_OBJECT must always be specified in the parameter area.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_MODIFY_OBJECT
1.	obj_type	Object type / parameter type
2.	obj_number	1 / 0
3.	id_lth	Length of object name in identification area / 0
	select_lth	0
4.	data_lth	Length of data structure in data area

Identification area

5.	Object name / —

Selection area

—

Data area

6.	Data structure of object type or parameter type / —

KDCADMI call

KDCADMI (&parameter_area, &identification_area, NULL, &data_area) or KDCADMI (&parameter_area, NULL, NULL, &data_area) or KDCADMI (&parameter_area, NULL, NULL, NULL)

Data returned by UTM

Parameter area (from page 385)

	Field name	Content
7.	retcode	Return codes

1. In the *obj_type* field you specify the type of object whose properties are to be modified or the type of application parameters which are to be modified. The following modifications are permissible:

   Object types

   – KC_CLUSTER_NODE
   (only possible in a UTM cluster application)
   Specify if you want to modify the computer names and/or filebase names of a node application.
   You must, for example, specify KC_CLUSTER_NODE if you want to assign actual values for the computer name of the node and the base name of the node application's KDCFILE to a reserve node application (see openUTM manual "Generating Applications" and openUTM manual "Using openUTM Applications").

   – KC_KSET
   Specify if you want to change keys in a key set.

   – KC_LOAD_MODULE
   Specify if you want to replace load modules of a UTM application on BS2000 systems or shared objects/DLLs of a UTM application on a Unix, Linux or a Windows system, i.e. if you want to load another version of a load module/shared object/DLL.

   – KC_LPAP
   Specify if you want to perform an operation for an LPAP partner of the application, i.e. if you want to modify the logical properties of an LU6.1 partner application.

   – KC_LSES
   Specify if you want to modify the properties of a session with an LU6.1 partner application.

– KC_LTAC
Specify if you want to modify the local properties of a remote service, i.e. the properties of an LTAC.

– KC_LTERM
Specify if you want to modify the properties of an LTERM partner.

B
B – KC_MUX
Specify if you want to modify the properties of a multiplex connection.

– KC_OSI_CON
Specify if you want to modify the properties of the connections to an OSI TP partner application.

– KC_OSI_LPAP
Specify if you want to perform an operation for an OSI-LPAP partner, i.e. you want to modify the logical properties of an OSI TP partner application.

– KC_PTERM
Specify if you want to perform operations for terminals, printers, client applications or TS applications.

– KC_TAC
Specify if you want to modify the properties of a transaction code which is assigned to a local service or a TAC queue.

– KC_TACCLASS
Specify if you want to modify the maximum number of processes that can process jobs concurrently for a certain TAC class.

– KC_TPOOL
Specify if you want to modify the properties of the LTERM partner or the number of active LTERM partners of an LTERM pool.

– KC_USER
Specify if you want to modify the properties of a user ID or its queue.

Parameter types

– KC_CLUSTER_CURR_PAR
Specify if you want to reset the statistics values of the cluster page pool in a UTM cluster application.

– KC_CLUSTER_PAR
Specify if, for a UTM cluster application, you want to

  – modify the parameters which control the way the individual node applications interact to check their availability.
  – modify the parameters which control node application accesses to the cluster configuration file and the cluster administration journal.

- KC_CURR_PAR
  Specify if you want to reset application-specific statistical values.
- KC_DIAG_AND_ACCOUNT_PAR
  Specify if you want to activate or deactivate diagnostic functions or if you want to modify the UTM accounting settings.
- KC_MAX_PAR
  Specify if you want to modify maximum values for applications (the MAX parameter) or, in UTM(BS2000) applications, if you want to activate or deactivate the supply of data to openSM2.
- KC_TASKS_PAR
  Specify if you want to modify values relating to the number of application processes, i.e. the total number of processes, maximum number of processes for executing asynchronous jobs etc.
- KC_TIMER_PAR
  Specify if you want to modify timer settings.

Point 6 on page 319f states which modifications are possible for each object type and parameter type.

2. What you have to specify in the *obj_number* field is determined by what is entered in the *obj_type* field:
   - specify *obj_number*=1 when you specify an object type in *obj_type* (exception: KC_TACCLASS, see below).
   - specify *obj_number*=0 when you specify a parameter type in *obj_type* or if you want to reset values in *obj_type* = KC_TACCLASS for all TAC classes.
3. What you have to specify in the *id_lth* field is determined by what is specified in the *obj_type* field:
   - if you specify an object type in *obj_type*, you must specify the length of the data structure in *id_lth* which you pass to UTM in the identification area.
     Exception: If *obj_type* = KC_TACCLASS you must specify *id_lth*=2.
   - if you specify a parameter type in *obj_type*, you must set *id_lth*=0.
4. In the *data_lth* field you specify the length of the data structure which you are passing to UTM in the data area.

   data_lth=0 is not permitted.

5. In the identification area you pass to UTM the name of the object whose properties you want to modify. This means that:

   – If you specify an object type in *obj_type*, then, in the identification area, you must pass the complete name of the object to UTM. Section 6 specifies for each object type the information you must state in the identification area.
   Exception: If *obj_type* = KC_TACCLASS and you reset values for all TAC classes then you must enter binary 0.

   – If you specify a parameter type in *obj_type*, then you do not need to pass any identification area to UTM. UTM ignores any information specified in the identification area.

6. In the data area you pass the data structure of the object or parameter type specified in *obj_type*. Each individual object or parameter type has its own data structure, which you must assign via the data area. You must pass the new property or parameter values to UTM in the data structure. You must complete the remaining fields of the data structure, i.e. the property or parameter value fields, which you do not wish to or cannot modify with binary zero before the call.

   X X X X In openUTM on Unix or Linux systems, it is not always necessary to pass data in the data area for *obj_type* = KC_LOAD_MODULE since, when transferring shared objects without any version specification, the name of the shared object in the identification area is sufficient.

   The following tables as of page 320 describe the modifications that are permitted as a function of object type/parameter type. You will be able to see from the description which properties/parameters you are able to modify and how the fields are to be completed. All the data structures are described in section “Data structures used to pass information” on page 443.

7. UTM writes the return code for the call to the *retcode* field, see “Return codes” on page 385.

11.2.9.1 obj_type=KC_CLUSTER_NODE

The modifications relate to a node application in a UTM cluster application.

In the identification area, you must specify the internal number in the cluster (index of the entry for this node in KC_GET_OBJECT for the object KC_CLUSTER_NODE) of the node application (field *kc_name2* in union *kc_id_area*). In the data area, you must pass the data structure *kc_cluster_node_str* with the new property values. You can only modify nodes that are not active.

Enter the following in the data structure *kc_cluster_node_str*:

	Field name	Meaning
	hostname	*hostname* contains the primary host name of the node on which this node application is running. *hostname* can be up to 8 characters in length.

B
B
B
B
B
X/W
X/W
X/W
X/W
	filebase	Base name of the KDCFILE, the user log file and the system log file SYSLOG for the node application. When the node application is started, the UTM system files are expected under the name specified here. This file structure must be accessible from all node applications.
		The name is passed in the element *filebase* of type *kc_file_base*:
		struct kc_file_base
		`char length[2];`
		`char fb_name[42];`
		fb_name Base name
		length Length of the base name
		Please note the following when modifying the base name of a node application:
		– The base names of the individual node applications of a UTM cluster application must differ from each other.
		– *BS2000 systems:*
		Specify the name without catalog ID. You must specify the catalog ID in the operand *catid_A*.
		The base name may contain a BS2000 user ID and be up to 42 characters in length.
		– *Unix, Linux and Windows systems:*
		Specify the directory which contains the UTM system files for the node applications. The name specified here must identify the same directory for all the nodes. It may be up to 27 characters in length.

B B	catid_A	Catalog ID assigned to the UTM system files of the node application (in particular the KDCFILE).

Field name	Meaning
virtual_host_name	In UTM cluster applications, this has the same function as the HOSTNAME parameter in the MAX generation statement. You may not specify MAX HOSTNAME in UTM cluster applications. BS2000 systems: Name of the virtual host on which the node application is to run from the perspective of BCAM. Unix or Linux systems: Specifying *virtual_host_name* permits the specification of the sender address for network connections established from this node application.

Period of validity / transaction management: type GID (page 313)

The effect is permanent. The information is stored in the cluster configuration file. The modification takes effect immediately and cannot be undone by rolling back the transaction.

11.2.9.2 obj_type=KC_KSET

The changes apply to the keys (key/access codes) of a key set.

In the identification area you must specify the name of the key set (*kc_name8* field of the *kc_id_area* union). In the data area you must pass the *kc_kset_str* data structure with the new property values.

Possible modification

With the exception of the MASTER key set, you can change one or more keys in a key set. The key set must exist in the configuration of the application.

Specify the following in the *kc_kset_str* data structure:

Field name	Meaning
keys[4000]	A key or access code is an integer between 1 and the value KEYVALUE, which was specified in the MAX statement at KDCDEF generation. *keys* consist of 4000 field elements (*keys*[0] to *keys*[3999]). The contents of the field elements are to be interpreted as follows: *keys*[0]= ´0´: The key/access code 1 does not belong to this key set. ´1´: The key/access code 1 belongs to this key set. *keys*[n]= ´0´: The key/access code n+1 does not belong to this key set. ´1´: The key/access code n+1 belongs to this key set. *keys*[3999]= ´0´: The key/access code 4000 does not belong to this key set. ´1´: The key/access code 4000 belongs to this key set.

Period of validity/ transaction management: type GPD (page 313)

11.2.9.3 obj_type=KC_LOAD_MODULE

This operation relates to a load module (BS2000) or to a shared object or DLL (Unix, Linux and Windows systems).

You must pass the name of the load module/shared object to UTM in the identification area (field *kc_name32* of union *kc_id_area*).

You must pass the data structure *kc_load_module_str* in the data area.

Possible modification

You can replace a load module, a shared object or a DLL in an application program or mark a load module in the common memory pool (BS2000 systems) for replacement.

The specified load module/shared object/DLL must exist in the application configuration, i.e. it must have been statically generated with KDCDEF. Specify the following in the data structure *kc_load_module_str*:

	Field name	Meaning
B B B B B B B B B	version[24]	Pass in *version* the version of the load module or shared object to be loaded. In UTM applications on BS2000 systems, you must always specify the version of the load module to be loaded. When the replacement is initiated, the library assigned to the load module during KDCDEF generation (see also lib in *kc_load_module_str*, page 475), an element with the name specified in the identification area and the version specified in *version* must all be available. In UTM cluster applications, this applies for all node applications. You cannot replace load modules that have the STATIC load mode (*load_mode*=´S´). Neither can load modules with the STARTUP load mode (*load_mode*=´U´) and which contain TCB entries be replaced.
X/W X/W X/W X/W		In UTM applications on Unix, Linux or Windows systems, you must specify the version if the shared object/DLL is generated with ONCALL load mode (*load_mode*=´O´). In the case of shared objects/DLLs with STARTUP load mode (*load_mode*=´U´), specifying the version is optional if you are not using the version concept.

Period of validity/ transaction management: type GID (page 313)

How replacement is made is determined by the load mode of the load module/shared object/DLL (field *load_mode* in *kc_load_module_str*, see page 475):

– *load_mode*=´U´ (STARTUP)

  The replacement is executed for each process before the next job is processed, without the current application program being terminated. Several application processes can be replaced simultaneously. You cannot initiate any further replacements until program replacement has been completed by all application processes.

– *load_mode*=´O´ (ONCALL)

The replacement is performed for each process only when a program unit from this load module/shared object/DLL is next called in this process. Replacement can be performed simultaneously be several processes.

B – *load_mode*=´P´, ´T´, ´C´ (POOL, POOL/STARTUP, POOL/ONCALL)

B In standalone UTM applications,
B a call to KC_MODIFY_OBJECT does **not** result in the replacement of the load module.
B Instead, the new version of the load module is marked.

B You must explicitly request the replacement of the load module by calling
B KC_CHANGE_APPLICATION or by restarting the application. By using several
B KC_MODIFY_OBJECT calls, you can mark several load modules which are then
B replaced when KC_CHANGE_APPLICATION is next invoked. If no
B KC_CHANGE_APPLICATION call is made in the same application run, the marked
B versions are then replaced when next the application is started.

B If you issue a KC_GET_OBJECT call between the KC_MODIFY_OBJECT call and the
B KC_CHANGE_APPLICATION call, then the marked version is already output as the
B current version, even if it has not yet been loaded. The KC_MODIFY_OBJECT call
B ensures that the new version of the load module is entered in the UTM tables as the
B current version and the currently loaded version is entered as the preceding version.
B You can tell from the *change_necessary* field whether a program replacement with
B KC_CHANGE_APPLICATION is still necessary in order to load the specified version.

B In UTM cluster applications,
B the version of the load module is modified and the replacement of the entire application
B program is initiated. The call causes the application to be unloaded and then immedi-
B ately reloaded again. The application that is loaded in this case contains the new
B versions of the load modules that were marked with KC_MODIFY_OBJECT.

C KDCPROG (page 764)

11.2.9.4 obj_type=KC_LPAP

These operations relate to an LPAP partner, i.e. to the logical properties of an LU6.1 partner application or to the connection to this partner application.

You must specify the name of the LPAP partner in the identification area (field *kc_name8* of the union *kc_id_area*). This is the name that was defined during KDCDEF generation in the LPAP statement for the partner application. In the data area you must pass the data structure *kc_lpap_str* with the new values of the properties.

Possible modifications

- Disable an LPAP partner or release a disabled LPAP partner.

  It is no longer possible to establish a connection to the partner application through a disabled LPAP partner.

  Specify the following in the data structure *kc_lpap_str*.

Field name	Meaning
state=´N´	The LPAP partner is to be disabled. There must be no connection to the partner application in existence at the time the partner is disabled. You must shut down existing connections before disabling the partner with *connect_mode*=´N´ or *quiet_connect*=´Y´. It is not possible to shut down the connection and disable the LPAP partner in a single call as shutting down the connection may take a relatively long time.
state=´Y´	The LPAP partner is to be released, i.e. any existing lock is to be cancelled.

  Period of validity/ transaction management: type GPD (page 313)

- Activate or deactivate automatic connection setup.

  Automatic connection setup means that, whenever the application starts, UTM attempts to establish a connection to the partner application.

  If automatic connection is defined in both applications (the local application and the partner application), the connection between the two of them is established automatically as soon as they are both available.

  Specify the following in the data structure *kc_lpap_str*:

Field name	Meaning
auto_connect=´Y´	As of the next application start, UTM is to attempt to establish the connection to the partner application automatically whenever it starts.
auto_connect=´N´	As of the next application start, the connection to the partner application is no longer to be established automatically.

  Period of validity/ transaction management: type GPD (page 313)

- Change the period of time for which UTM monitors the idle state of a session to the partner application; i.e. if the session is not occupied by a job, UTM waits for this period of time before shutting down the connection.

  Specify the following in the data structure *kc_lpap_str:*

Field name	Meaning
idletime_sec[5]	Specify in *idletime_sec* the time in seconds for which UTM is to monitor the idle state of a session with the partner application *idletime_sec* = ´0´ means that the idle state is not monitored. Maximum value: ´32767´ Minimum value: ´60´, In the case of values that are smaller than 60 but not equal to 0 then the value 60 is used.

  Period of validity/ transaction management: type GPD (page 313)

  The timer modification only takes effect when the session next reaches the idle state, but not before the end of the program unit run (PEND) in which the call is processed.

- Set up or shut down the connection to the partner application.

  The connection can be shut down in two ways:

  – The connection can be shut down immediately, i.e. UTM shuts down the connection irrespective of whether or not jobs are currently being processed via the connection (*connect_mode*).

  – You can set the connection to QUIET (*quiet_connect*). QUIET means that UTM shuts down the connection to the partner application as soon as the sessions generated for the LPAP partner are no longer occupied by jobs (dialog or asynchronous jobs).

    However, no new dialog jobs are accepted for the LPAP partner. New asynchronous jobs are accepted, but no longer sent; they remain in the output queue.

Field name	Meaning
connect_mode=´Y´	UTM is to establish the connection to the partner application. If the LPAP partner is disabled, it must be released in a separate transaction before the connection is established (*state*=´Y´).
connect_mode=´N´	The connection to the partner application is to be shut down immediately. If the connection is shut down with *connect_mode* = ´N´, it is possible that services or conversations may be aborted abnormally. It is better to shut down the connection with *quiet_connect* = ´Y´.

  Period of validity / transaction management: type A (page 313)

Field name	Meaning
quiet_connect=´Y´	The property QUIET is set for the connection to the partner application. The property QUIET can be reversed with *connect_mode*=´Y´.

Period of validity / transaction management: type IR (page 313)

The fields *connect_mode* and *quiet_connect* cannot be set simultaneously within a call. Moreover, *connect_mode*=´Y´ cannot be set simultaneously with *state* =´N´. If a KC_MODIFY call with *connect_mode*=´N´ is transmitted for a connection which has been set to QUIET, the connection is then shut down immediately.

connect_mode=´N´ “overwrites” *quiet_connect*=´Y´.

- Activate or deactivate the BCAM trace for the connection to the partner application.

  The precondition for LPAP-specific activation is that the BCAM trace is not generally activated, i.e. the trace is either completely deactivated or is only explicitly activated for selected LTERM/LPAP partners or USERs.

  The precondition for LPAP-specific deactivation is that the BCAM trace can be deactivated for a specific LPAP partner only if the BCAM trace is not generally activated.

  You will find information about the general activation and deactivation of the BCAM trace in the description of the data structure *kc_diag_and_account_par_str* starting on page 610.

Field name	Meaning
bcam_trace=´Y´	The BCAM trace is specifically activated for this LPAP partner. Events are logged on all transport connection to the partner application assigned to this LPAP partner. When the trace function is activated, each application process creates its own trace file.
bcam_trace=´N´	The BCAM trace is explicitly deactivated for this LPAP partner. The trace files are closed only when the trace function is deactivated generally (object type KC_DIAG_AND_ACCOUNT_PAR; page 369).

Period of validity / transaction management: type IR (page 313)

KDCLPAP (page 743) / KDCDIAG (on page 681) for the BCAM trace

11.2.9.5 obj_type=KC_LSES

This modification relates to a session for distributed processing using the LU6.1 protocol.

In the identification area you must pass the session name (LSES name from KDCDEF generation) to UTM (*kc_name8* in the union *kc_id_area*).

In the data area you must pass the data structure *kc_lses_str* with the new values of the properties.

Possible modifications

- Establish a transport connection to the partner application for the session.

Field name	Meaning
connect_mode=´Y´ con, pronam, bcamappl	A transport connection is to be established for the session. If a specific transport connection is to be established for a session, then you must unambiguously specify this transport connection in *con*, *pronam*, *bcamappl*. To do this, you must specify the following information: – in *con*, the name of the connection defined at creation or generation of the CON object – in *pronam* the name of the computer on which the partner application is running – in *bcamapp* the name of the local UTM application (BCAMAPPL name) through which the connection to the partner application is established. If you do not specify *con*, *pronam*, *bcamappl*, then UTM establishes any of the transport connections configured dynamically or generated for the partner application with the KDCDEF control statement CON. A connection cannot be established if the associated LPAP partner is disabled (see KC_LPAP *state* =´N´ on page 325). If the LPAP partner is disabled, it must be released with an explicit KC_MODIFY_OBJECT call before the connection is established (KC_LPAP with *state*=´Y´).

Period of validity / transaction management: type A (page 313)

- Shut down the transport connection that exists for the session.

  You can instruct UTM to shut down the connection immediately or you can assign the property QUIET to the connection. QUIET means that UTM shuts down the connection to the partner application as soon as the session is no longer occupied by jobs (dialog or asynchronous jobs). No further new dialog jobs are accepted. New asynchronous jobs are accepted, but no longer sent; they remain in the output queue.

Field name	Meaning
connect_mode=´N´	The connection to the partner application that exists for the session is to be shut down immediately. Shutting down the connection with *connect_mode* = ´N´ takes immediate effect, with the result that services or conversations may be terminated abnormally. It is better to shut down the connection with *quiet_connect* = ´Y´.

  Period of validity / transaction management: type A (page 313)

Field name	Meaning
quiet_connect=´Y´	Set the property QUIET for the connection to the partner application. The property QUIET is cancelled with *connect_mode*=´Y´.

  Period of validity / transaction management: type IR (page 313)

  No other field in the data structure can be occupied at the same time as *connect_mode*=´N´. In particular, *connect_mode* and *quiet_connect* cannot be set simultaneously.

  If a connection which has previously been set to QUIET is now set to *connect_mode*=´N´, the connection is shut down immediately. The property QUIET is overwritten by *connect_mode*=´N´.

  KDCLSES (page 751)

11.2.9.6 obj_type=KC_LTAC

This modification relates to an LTAC, i.e. to a local application transaction code for a service in a partner application.

You must pass the name of the LTAC to UTM in the identification area (*kc_name8* in the union *kc_id_area*).

In the data area you must pass the data structure *kc_ltac_str* with the new values of the properties.

Possible modifications

- You can modify the maximum time which UTM will wait to access a session when requesting a remote service. To do this, specify the following in *kc_ltac_str*:

Field name	Meaning
accesswait_sec[5]	Specify in *accesswait_sec* the time in seconds which UTM at most is to wait after the LTAC call to reserve a session or to establish an association. When specifying the time, you should remember that the actual transport connection to the partner application may still have to be established. In asynchronous LTACs, *accesswait_sec* ≠ 0 means that the job is always entered in the local message queue for the partner application. Wait time *accesswait_sec*=0 means: In dialog LTACs, the local service that is calling the remote service is immediately continued with the appropriate return code if no session or association to the partner application is free or if the local application is the “contention loser” (see *kc_lpap_str* page 478f; field *contwin*). In asynchronous LTACs, the asynchronous job is rejected with a return code at the FPUT call if no connection to the partner application exists. If there is a connection to the partner application, the message is entered in the message queue. Minimum value: ´0´; maximum value: ´32767´

Period of validity / transaction management: type GPR (page 313)

- You can modify the maximum time which UTM will wait for a reply from a remote service. To do this, specify the following in *kc_ltac_str*:

Field name	Meaning
replywait_sec[5]	Specify in *replywait_sec* the maximum time in seconds which UTM is to wait for a reply from the remote service. By limiting the waiting time, it can be ensured that users do not have to wait indefinitely at the terminal. *replywait_sec* = ´0´ means: wait without a time limit. Minimum value: ´0´; maximum value: ´32767´

Period of validity / transaction management: type GPR (page 313)

- You can disable the LTAC or release it again. Disabling an LTAC means that no further jobs are accepted from the local application for the remote service to which the LTAC is assigned. To do this, specify the following in *kc_ltac_str*:

Field name	Meaning
state=´N´	The LTAC is to be disabled, UTM is to accept no further jobs for the associated remote service.
state=´Y´	The (disabled) LTAC is to be released, i.e. the lock is to be cancelled.

Period of validity / transaction management: type GPR (page 313)

KDCLTAC (page 753)

11.2.9.7 obj_type=KC_LTERM

This modification relates to an LTERM partner.

You must pass the name of the LTERM partner to UTM in the identification area (*kc_name8* in the union *kc_id_area*).

In the data area you must pass the data structure *kc_lterm_str* with the new values of the properties.

Possible modifications

- Disable the LTERM partner or release the disabled LTERM partner. LTERM partners in an LTERM pool cannot be disabled or released with *obj_type*=KC_LTERM (see in this connection *obj_type*= KC_TPOOL; page 355).

  To disable or release an LTERM partner, specify the following in *kc_lterm_str*:

Field name	Meaning
state=´N´	Disables the LTERM partner. Disabling a dialog partner (*usage_type*=´D´) has the following effect: – A client connection request is performed. The connection is disabled and UTM message K027 is output. With the exception of KDCOFF, no client/user jobs are performed. – Any existing connection is maintained. Any input with the exception of KDCOFF is acknowledged with UTM message K027. The lock does not take effect until a synchronization point (end of transaction) is reached on this connection. If the LTERM partner is disabled, KDCOFF BUT has the same effect as KDCOFF. If the LTERM partner of a printer is disabled, the print jobs are retained in the message queue. Print jobs initiated after a disable operation are not rejected; they are entered in the message queue.
state=´Y´	Releases the LTERM partner, i.e. cancels a lock.

  Period of validity / transaction management: Type GPD (page 313)

- Set up or shut down the connection to the client or printer assigned to this LTERM partner.

Field name	Meaning
connect_mode=´Y´	The connection to the client/printer is to be set up. *connect_mode*=´Y´ is not permitted if the LTERM partner you have specified in the identification area belongs to an LTERM pool or is assigned to a UPIC client.

Field name	Meaning
connect_mode=´N´	The connection to the client/printer is to be shut down immediately. A connection shutdown initiated with *connect_mode* = ´N´ takes immediate effect, with the result that services may be terminated abnormally (PEND ER). Using *connect_mode*=´N´, you can also shut down the connection to a client that is connected to the application via an LTERM pool, i.e. you can also specify in the identification area the name of an LTERM partner that belongs to an LTERM pool.

Period of validity / transaction management: type A (page 313)

- Assign a new start format to the LTERM partner or delete the start format of the LTERM partners.

You can assign a start format to each LTERM partner that has been configured for connecting terminals. In order to modify the start format, you must always specify the format name and the format attribute of the new start format.

A precondition for allocation of a start format is that a formatting system must have been generated (KDCDEF statement FORMSYS). If the start format is a #format, then a sign-on service must also have been generated.

Field name	Meaning
format_attr	Format identifier for the new start format: ´A´ for the format attribute ATTR. The format name at the KDCS program interface is +*format_name*. ´N´ for the format attribute NOATTR. The format name at the KDCS program interface is ∗*format_name*. ´E´ for the format attribute EXTEND. The format name at the KDCS program interface is #*format_name*. The meanings of the format attributes are described on page 495.
format_name[7]	Name of the start format. The name may be up to 7 characters long and may contain only alphanumeric characters.

To delete the start format, enter blanks in *format_attr* and *format_name*.

Period of validity / transaction management: type GPD (page 313)

- Activate the BCAM trace for the connections for this LTERM partner.

  The BCAM trace function monitors all connection-related activity.

  The precondition for LTERM-specific activation is:

  The BCAM trace is not generally activated for all LTERM and LPAP partners, i.e. the trace is either completely deactivated or explicitly activated only for selected LTERM/LPAP partners and USERs.

  The precondition for LTERM-specific deactivation is:

  The BCAM trace can only be deactivated for specific LTERM partners if the BCAM trace is not generally activated.

  You will find information about general activation and deactivation of the BCAM trace in the description of the data structure *kc_diag_and_account_par_str* starting on page 610.

Field name	Meaning
bcam_trace=´Y´	The BCAM trace is explicitly activated for this LTERM partner. All events on the connection to the client/printer assigned to this LTERM partner are logged. When the trace function is activated, each application process creates its own trace file.
bcam_trace=´N´	The BCAM trace is explicitly deactivated for this LTERM partner. The trace files are closed only when the trace function is deactivated generally (object type KC_DIAG_AND_ACCOUNT_PAR; page 369).

  Period of validity / transaction management: type IR (page 313)

- Exchange the master LTERMs of two LTERM bundles or add a group LTERM to a different LTERM group.

  This function is only permitted in standalone UTM applications.

  If the LTERM is the master LTERM of the LTERM bundle, you can replace all the slave LTERMs and the associated PTERMs with a different master LTERM. In this event, a master LTERM of an LTERM bundle must be specified in the *master* parameter.

  If the LTERM is a group LTERM of an LTERM group, you can assign it to a different LTERM group. The primary LTERM that you specify in the *master* parameter must either be a normal LTERM, a primary LTERM of an LTERM group or a master LTERM of an LTERM bundle. A normal LTERM must fulfill the following conditions:

  – A PTERM with the PTYPE APPLI or SOCKET must be assigned to the LTERM.
  – The LTERM must not be a slave LTERM of an LTERM bundle.
  – The LTERM must have been generated with USAGE=D.

Specify the following in the data structure *kc_lterm_str*:

Field name	Meaning
master[8]	The name of a master LTERM in an LTERM bundle, the name of a primary LTERM in an LTERM group or the name of the normal LTERM. The name can be up to 8 characters in length and may only contain alphanumeric characters.

Period of applicability / transaction management: type PD (page 313)

Some of the modifications can also be performed with KDCLTERM (page 755) or KDCDIAG (page 681).

11.2.9.8 obj_type=KC_MUX

This operation relates to a multiplex connection.

You must identify the multiplex connection unambiguously in the identification area. To do this, in the data structure *kc_triple_str* of the union *kc_id_area*, pass the name of the multiplex connection, the name of the computer on which the associated message router is located, and the name of the UTM application through which the multiplex connection is to be established.

In the data area you must pass the data structure *kc_mux_str* with the new values of the properties.

Possible modifications

- Disable a multiplex connection or release a disabled multiplex connection.

  No connection between the message router and the UTM application can be set up via a disabled multiplex connection. Specify the following in the data structure *kc_mux_str*:

Field name	Meaning
state=´N´	Disables a multiplex connection There must be no current connection to the multiplex connection. You must shut down any existing connections with *connect_mode*=´N´. It is not possible to shut down the connection and disable a multiplex connection in a single KC_MODIFY_OBJECT call as shutting down the connection can take some time.
state=´Y´	Releases a multiplex connection, i.e. cancels a lock.

  Period of validity / transaction management: type GPD (page 313)

- Increase or reduce the maximum number of clients that can be connected concurrently via this multiplex connection.

Field name	Meaning
maxses[5]	Specify in *maxses* the maximum number of sessions that can exist between the message router and the application. Minimum value:´1´; Maximum value:´65000´ (theoretical value)

  Period of validity / transaction management: type GPR (page 313)

- Activate or deactivate automatic connection setup to the multiplex connection.

  In automatic connection setup, UTM attempts to establish a connection to the multiplex connection automatically whenever the application starts.

Specify the following in the data structure *kc_mux_str*.

Field name	Meaning
auto_connect=´Y´	As of the next application start, UTM is to attempt to establish the connection to the multiplex connection automatically.
auto_connect=´N´	As of the next application start, UTM is no longer to establish the connection the multiplex connection automatically. It must then be established explicitly by the administrator.

Period of validity / transaction management: type GPD (page 313)

- Set up or shut down the connection to the message router for the multiplex connection.

Specify the following in the data structure *kc_mux_str*:

Field name	Meaning
connect_mode=´Y´	UTM is to establish the connection to the message router. If a connection is to be established for a disabled multiplex connection, the multiplex connection must be released before connection setup with its own KC_MODIFY_OBJECT call (*state*=´Y´). *connect_mode* = ´Y´ cannot be set at the same time as *state* =´N´ (disable multiplex connection).
connect_mode=´N´	The connection to the message router is to be shut down immediately. A connection shutdown initiated with *connect_mode* = ´N´ takes immediate effect, so it is possible for sessions to be terminated abnormally.

Period of validity / transaction management: Type A (page 313)

- Activate or deactivate the BCAM trace for this multiplex connection. Specify the following in the *kc_mux_str* data structure:

Field name	Meaning
bcam_trace=´Y´	The BCAM trace is activated explicitly for this multiplex connection. All the events on the connection to the message router assigned to this multiplex connection are recorded. When the trace function is created, every process of the application generates its own trace file.
bcam_trace=´N´	The BCAM trace is deactivated explicitly for this multiplex connection. The trace files are not closed until the trace is deactivated with general validity (object type KC_DIAG_AND_ACCOUNT_PAR; page 369).

Period of validity / transaction management: type IR (page 313)

KDCMUX (page 758) / KDCDIAG (page 681) for the BCAM trace

11.2.9.9 obj_type=KC_OSI_CON

This operation relates to a connection for distributed processing via OSI TP.

In the identification area you must specify the name of the connection defined during KDCDEF generation in OSI-CON (field *kc_name8* of the union *kc_id_area*).

In the data area, you must specify the data structure *kc_osi_con_str* with the new values of the properties.

Possible modification

You can activate a replacement connection (connection set to inactive) to an OSI TP partner application. Specify the following in the data structure *kc_osi_con_str*:

Field name	Meaning
active=´Y´	UTM is to activate the replacement connection. Before UTM activates the replacement connection, UTM deactivates the previously active connection. No association to the related partner application may therefore be in existence when the replacement connection is activated.

Period of validity / transaction management: type GIR (page 313)

KDCLPAP (page 743) operand OSI-CON

11.2.9.10 obj_type=KC_OSI_LPAP

This operation relates to an OSI-LPAP partner, i.e. to the logical properties of an OSI TP partner application or to the connection to this partner application.

In the identification area you must specify the name of the associated OSI-LPAP partner (field *kc_name8* of the union *kc_id_area*). The name is defined during KDCDEF generation in the OSI-LPAP statement for the partner application.

In the data area you must pass the data structure *kc_osi_lpap_str* with the new values of the properties.

Possible modifications

If the OSI-LPAP is the master LPAP of an OSI-LPAP bundle, you can only modify the *state* field.

- Disable an OSI-LPAP partner or release a disabled OSI-LPAP partner.

  It is not possible to make a connection to the partner application via a disabled OSI-LPAP partner.

  Specify the following in the data structure *kc_osi_lpap_str*:

Field name	Meaning
state=´N´	The OSI-LPAP partner is to be disabled. There must be no current connection to the partner application at the time of the disable operation. You must shut down existing connections before disabling the partner, using a separate call with *connect_number*=´0´ or *quiet_connect*=´Y´. You cannot shut down the connection and disable the OSI-LPAP partner in a single transaction.
state=´Y´	The OSI-LPAP partner is to be released, i.e. there is a lock in existence which is to be cancelled.

  Period of validity / transaction management: type GPD (page 313)

- Increase or reduce the number of connections to the partner application which UTM automatically establishes when the application starts.

  In automatic connection setup, UTM attempts to establish the required number of connections to the partner application whenever the application starts.

  If automatic connection setup is defined in both applications (the local application and the partner application), the connection between the two of them is established automatically as soon as both applications are available.

Specify the following in the data structure *kc_osi_lpap_str*:

Field name	Meaning
auto_connect_number	Specify in *auto_connect_number* the number of connections to the partner application which UTM is to establish automatically when the application next (and subsequently) starts. The OSI-LPAP partner via which the partner application connects must not be disabled. If you specify *auto_connect_number* = ´0´, automatic connection setup does not occur when the application next starts. If a number is specified that is greater than the generated maximum number of parallel connections (see field *associations* in *kc_osi_lpap_str*), then, on the next start, UTM attempts to establish all generated parallel connections (= number in *associations*). The value specified in *auto_connect_number* must, however be less than or equal to ´32767´. Minimum value: ´0´. Maximum value: generated maximum number of parallel connections (*associations*)

Period of validity / transaction management: type GPD (page 313)

- Increase or decrease the number of parallel connections that should currently exist between the UTM application and the partner application; i.e. additional connections can be established or some of the existing connections can be shut down. Setting up additional connections is only possible if the maximum number of parallel connections to the partner application generated with KDCDEF has not already been established.

Specify the following in the data structure *kc_osi_lpap_str*:

Field name	Meaning
connect_number	Specify in *connect_number* the total number of connections to the partner application that should exist. The effect of the call is thus determined by what is specified for *connect_number*. Distinctions must be drawn between the following situations: – If you specify a number in *connect_number* which is less than the number of parallel connections that are currently established, UTM shuts down connections to the partner application until only *connect_number* connections are in existence. To begin with, UTM shuts down any connections that are not currently reserved by jobs. When this has been done, if there are still more connections open than the number specified in *connect_number*, then UTM begins to also shut down connections that are reserved by jobs. Any currently active services or conversations are aborted when this happens. If you specify *connect_number* = ´0´, UTM shuts down all connections to the partner application. – If you specify a number in *connect_number* which is greater than the number of parallel connections that are currently established, UTM attempts to establish further connections to the partner application until a total of *connect_number* connections are in existence. However, the maximum number of parallel connections which UTM will establish to the partner application is that established during KDCDEF generation for the OSI-LPAP partner belonging to the partner application. This maximum number is returned when information is requested in the *associations* field of *kc_osi_lpap_str*. In other words, if *connect_number* > *associations*, then UTM only establishes the generated maximum number of connections. If connections are to be established to a disabled OSI-LPAP partner, you must re-enable this partner beforehand (see *state* field on page 339). The OSI-LPAP partner must be released in a separate KC_MODIFY_OBJECT. *connect_number* and *quiet_connect* cannot be specified together in a single KC_MODIFY_OBJECT call. Likewise, *connect_number* must not be specified together with *state*=´N´. Minimum value: ´0´ Maximum value: the number returned by UTM in *associations*; a numeric value greater than ´32767´ will be rejected.

Period of validity / transaction management: type A (page 313)

- Shut down all parallel connections to the partner application.

  You can instruct UTM to shut down all connections immediately or to assign the property QUIET to the connections. QUIET means that UTM shuts down the connection to the partner application as soon as the partner application is no longer occupied by jobs (dialog or asynchronous jobs). No further new dialog jobs are accepted. New asynchronous jobs are accepted, but no longer sent; they remain in the output queue.

Field name	Meaning
connect_number=´0´	If you specify *connect_number* = ´0´, UTM shuts down all connections to the partner application. The connections are shut down even if there are active services or conversations on the connection. These are aborted. It is thus better to shut down connections with *quiet_connect* = ´Y´.

  Period of validity / transaction management: type A (page 313)

Field name	Meaning
quiet_connect=´Y´	The property QUIET is set for the connections to the partner application. The property QUIET can be reset with *connect_number* > ´0´.

  Period of validity / transaction management: type IR (page 313)

  connect_number and *quiet_connect* cannot be set concurrently within a single KC_MODIFY_OBJECT call.

- Modify the period of time for which the idle state of the UTM application association to the partner application is monitored. In other words, if the association is not occupied by a job, UTM waits for this period of time before UTM shuts down the connection.

  Specify the following in the data structure *kc_osi_lpap_str:*

Field name	Meaning
idletime_sec[5]	Specify in *idletime_sec* the time in seconds for which UTM is to monitor the idle state of an association to the partner application. *idletime_sec* = ´0´ means that the idle state it not monitored. Maximum value: ´32767´ Minimum value: ´0´, In the case of values that are smaller than 60 but not equal to 0 then the value 60 is used.

  Period of validity / transaction management: type GPD (page 313)

  The modification of the timer takes effect when the association next reaches the idle state, but not before the end of the program unit run (PEND) in which the call is processed.

KDCLPAP (page 743)

11.2.9.11 obj_type=KC_PTERM

This operation relates to a client or printer for the application.

You must identify the client/printer unambiguously in the identification area. To do this, in the data structure *kc_triple_str* of the union *kc_id_area*, pass the name of the client printer, the name of the computer on which it is located, and the name of the UTM application via which the connection is to be established.

In the data area you must pass the data structure *kc_pterm_str* with the new values of the properties.

Possible modifications

- Change the client/printer assignment to the LTERM partner.

  In this way you can modify the logical properties of the client/printer. In particular, you can use them to assign a printer to a printer pool or to a printer control LTERM. When the assignment is modified, neither the client/printer nor the LTERM partner to which the client/printer is assigned may be connected to the application.

  Restriction:

  Reassignment of the LTERM partner is possible only for terminals and printers. For UPIC clients, TS applications (APPLI/SOCKET) generated as dialog partners, and clients that connect to the application using an LTERM pool, it is not possible to change the assignment to an LTERM partner defined at configuration.

  When you assign a new LTERM partner to a terminal or printer, the LTERM partner must not be currently assigned or have been previously assigned to a client/printer of another protocol type. Distinctions are drawn here between the following four protocol types: terminals, TS applications, printers and RSO printers. It is not possible, for example,

  – to assign an LTERM partner that is or was assigned to a UPIC client or to a TS application to a terminal ,

  B
  B – to assign an LTERM partner that is or was assigned to a normal printer to an RSO printer (and vice-versa).

Field name	Meaning
lterm[8]	Specify in *lterm* the name of the LTERM partner that is to be assigned to this client/printer. This function is only permitted in standalone UTM applications. The LTERM partner must exist in the application configuration. It must not be an LTERM partner of an LTERM pool, a master or slave LTERM of an LTERM bundle or a group or primary LTERM of an LTERM group. The maximum length of the name is 8 characters. For clients, the old assignment of this LTERM partner is implicitly cancelled. Only printers that have been configured for output (*usage_type*=´O´) can be assigned to LTERM partners. For printers, the old assignment of LTERM partner specified in *lterm* is not cancelled if a printer was previously assigned to it. Both printers are combined into a printer pool. Any required number of printers may belong to a printer pool. If the LTERM partner is assigned to a printer control LTERM, the printer must have a printer ID which is unique in the printer control LTERM area, otherwise the call is rejected. *connect_mode* and *lterm* cannot be specified together in a single call.

Period of validity / transaction management: type PD (page 313)

- Activate or deactivate automatic connection setup to the client/printer.

  With automatic connection setup, UTM attempts to establish the connection to the client/printer automatically.

  Exception:

  Automatic connection setup cannot be achieved to clients which are connected to the application via an LTERM pool nor to UPIC clients. In both these cases, connection setup is always initiated by the client and not by the UTM application.

  Specify the following in the data structure *kc_pterm_str*:

Field name	Meaning
auto_connect=´Y´	As of the next application start, UTM is to establish the connection to the client/printer automatically, provided that the client/printer is available. The client/printer must not be disabled (*state*=´N´).
auto_connect=´N´	As of the next application start, UTM is no longer to establish the connection to the client/printer automatically.

Period of validity / transaction management: type GPD (page 313)

- Disable a client or printer or cancel an existing lock.

  You can disable only those clients and printers that have been entered explicitly and statically in the configuration, using a PTERM statement, or dynamically as an object of the type KC_PTERM. Clients which connect via an LTERM pool or a multiplex connection cannot be disabled.

  Specify the following in *kc_pterm_str* in order to disable or release a client/printer:

Field name	Meaning
state=´N´	Disable the client/printer. A lock on a client does not take effect until the client next attempts to establish a connection to the UTM application. The connection request is then rejected by UTM. Any connection that exists at the time of disable operation is maintained.
state=´Y´	The client/printer lock is to be cancelled.

  Period of validity / transaction management: type GPD (page 313)

- Set up or shut down t the connection to the client/printer.

Field name	Meaning
connect_mode=´Y´	The connection to the client/printer is to be established. Exception: *connect_mode*=´Y´ cannot be specified for clients which are connected to the application via an LTERM pool, nor for UPIC clients. The client/printer must not be disabled. A disabled client/printer must be released prior to setting up the connection (*state*=´Y´). Releasing the client/printer and setting up the connection cannot be performed in a single call.
connect_mode=´N´	The connection to the client/printer is to be shut down immediately. A connection shutdown initiated with *connect_mode* = ´N´ takes immediate effect. If the connection is occupied by a job at that time, processing of the job is aborted.
connect_mode=´R´	May only be specified for clients which are connected to a UTM application on a BS2000 system through a multiplex connection. *connect_mode*=´R´ (Release pending connections) instructs UTM to release a session in the DISCONNECT PENDING state once the timer has expired. The session cannot be released if the timer has not yet expired. See openUTM manual “Generating Applications” in relation to the DISCONNECT PENDING state.

  connect_mode and *lterm* cannot be specified to together in a single call.

  Period of validity / transaction management: type A (page 313)

- Change the maximum period for which UTM will wait for an entry from the client after the end of a transaction or after the sign-on. When the time is exceeded, the connection to the client is cleared down (only relevant in the case of dialog partners).

  Specify the following in the *kc_pterm_str* data structure:

Field name	Meaning
idletime[5]	In *idletime* you specify the maximum period in seconds for which openUTM waits outside a transaction (i.e. after the end of a transaction or after sign-on) for an entry from the client. When *idletime*=0 is specified, openUTM waits for an unlimited period. Maximum value: ´32767´ Minimum value: ´60´ In the case of values that are smaller than 60 but not equal to 0 then the value 60 is used.

  Period of validity / transaction management: type GPD (page 313)

  The modification of the timer takes effect at the next end of transaction but not before the end of the program unit run (PEND) in which the call is processed.

KDCPTERM (page 769) with the exception of *idletime*

11.2.9.12 obj_type=KC_TAC

This operation relates to a local service transaction code (*tac_type*='A' or 'D') or a TAC queue (*tac_type*='Q').

In the identification area, you must pass the name of the transaction code or TAC queue (field *kc_name8* of the union *kc_id_area*). In the data area, you must pass the data structure *kc_tac_str* with the new values of the properties.

You can change the status and data access control for transaction codes and TAC queues. For transaction codes you can also reset TAC-specific statistics values to 0. Statistics values cannot, however, be changed in a KC_MODIFY_OBJECT call.

Possible modification

- Modifying the status of a transaction code or TAC queue.

  You can either disable a transaction code or TAC queue or enable a disabled transaction code or TAC queue again.

  The administration command KDCTAC cannot be disabled.

  If you change the status of a transaction code in a call, the statistics values cannot be reset.

  Specify the following in *kc_tac_str* to disable or release the transaction code:

Field name	Meaning
state=´N´	The transaction code/TAC queue is to be disabled. Lock means that UTM will accept no further jobs for this transaction code or TAC queue. – *tac_type*=´A´ or ´D´: The transaction code is disabled as a service TAC (1st TAC of a service). It is not disabled as a follow-up TAC in a service (call type=´B´). Asynchronous jobs which are in the transaction code's message queue at the time of disabling are still started. – *tac_type*=´Q´: The TAC queue is disabled for write accesses; read accesses are possible. You cannot use *state*=´N´ to disable transaction codes for which *call_type*= ´N´ is set.

Field name	Meaning
state=´H´	The transaction code or TAC queue is to be completely disabled (Halt). – *tac_type*=´A´ or ´D´: The transaction code is disabled both as a service TAC and as a follow-up TAC in an asynchronous or dialog service. Asynchronous jobs which are in the transaction code's message queue at the time of the disable operation are not started. They remain in the queue until the transaction code is released again or is set to *state*=´N´. – *tac_type*=´Q´: The TAC queue is disabled for write and read accesses.
state=´K´	This state may only be specified for asynchronous transaction codes (*tac_type*=´A´) that are also service TACs (*call_type=´B´ or* ´F´) and for TAC queues. The transaction code or TAC queue is disabled. – *tac_type*=´A´: Jobs for the transaction code are accepted, but they are not processed. They are merely entered into its job queue. They are not processed until you change the status of the transaction code to ´Y´ or ´N´. – *tac_type*=´Q´: The TAC queue is disabled for read accesses; write access is still possible. You can use *state*=´K´ (Keep) to collect jobs that are not to be processed until such time as the load on the application is reduced (e.g. at night).
state=´Y´	The transaction code or TAC queue is to be released again. *state*=´Y´ resets both *state*=´N´, *state=´*H´ and *state*=´K´.

Period of validity / transaction management: type GID (page 313)

If the transaction code KDCMSGTC is disabled, then all UTM messages having a UTM message destination MSGTAC and which are still located in the page pool are deleted.

KDCTAC (page 789)

- Resetting statistical information for the transaction code to 0.

  You can reset the statistics values to 0 during a run by entering 0 in one of the following fields in *kc_tac_str*. UTM will then reset all fields to 0. A value ≠ 0 is rejected.

Field name	Meaning
used	Number of program unit runs with this transaction code
number_errors	Number of program unit runs which were terminated with errors.
db_counter	Average number of database calls from program units started using this transaction code.
tac_elap_msec	Average runtime of program units started using this transaction code (elapsed time)

Field name	Meaning
db_elap_msec	Average time needed to process database calls with this TAC in the program units.
taccpu_msec	Average CPU time in milliseconds needed to process this transaction code in the program unit. The value corresponds to the CPU time used by UTM and by the database system.
taccpu_micro_sec	Average CPU time in microseconds taken to process this transaction code in the program unit. This corresponds to the CPU time consumed by UTM plus the CPU time required by the database system.
nbr_ta_commits	Number of program unit runs for this TAC that have successfully concluded a transaction.
number_errors_ex	See *number_errors*.

You can either reset the statistics values for a specific transaction code or for all transaction codes in the application. If you want to reset the values for a specific transaction code you must enter the name of the transaction code in the identification area. In all other cases you must supply the identification area with binary zero.

Period of validity / transaction management: type GIR (page 313)

- You can modify the data access control for a transaction code. If the transaction code was protected up to now by a lock code, you can remove the lock code and control data access by means of an access list. The reverse also applies. Please note that a lock code and access list are mutually exclusive; only one type of data access control is permitted at any one time.

Field name	Meaning
lock_code[4]	*lock_code* can be a number between ´0´ and the upper limit defined in the MAX statement (KEYVALUE operand). ´0´ removes data access control.
access_list[8]	In *access_list* you can specify an existing key set or fill the field with blanks. Blanks remove the data access control.

A user can only access the transaction code when the key set of the user and the key set of the LTERM partner by means of which the user is signed on contain at least one key code that:

– corresponds to the lock code or

– is also contained in the key set specified in *access_list*

Period of validity / transaction management: type GPD (page 313)

- You can modify the data access control for a ZAC queue. Specify the following in the *kc_tac_str* data structure:

Field name	Meaning
q_read_acl[8]	In *q_read_acl* you specify the name of an existing key set by means of which the queue is protected against unauthorized reading and deletion. You can also remove the protection by specifying blanks. In this case, all users can read and delete messages from this queue.
q_write_acl[8]	In *q_write_acl* you specify the name of an existing key set by means of which the queue is protected against unauthorized write accesses. You can also remove the protection by specifying blanks. In this case, all users can write messages to this queue.

A user only has read (delete) access or write access to this TAC queue if the key set of the user and the key set of the logical terminal by means of which the user is signed on each contain at least one key code that is also contained in the specified key set.

Period of validity / transaction management: type GPD (page 313)

- Specify that queued messages are to be stored in the dead letter queue (TAC queue KDCDLETQ). Specify the following in the data structure *kc_tac_str*:

Field name	Meaning
dead_letter_q='Y'	Messages to this asynchronous TAC or this TAC queue which could not be processed are backed up in the dead letter queue if they are not redelivered and (with message complexes) no negative acknowledgement job has been defined. *dead_letter_q*='Y' is not permitted for KDCDLETQ, KDCMSGTC, all interactive TACs and asynchronous TACs with CALL=NEXT.
dead_letter_q='N'	Messages to this asynchronous TAC or this TAC queue which could not be processed are not backed up in the dead letter queue but deleted. This value must be specified for all interactive TACs and for asynchronous TACs with CALL=NEXT, as well as for KDCMSGTC and KDCDLETQ.

Period of validity / transaction management: type GPD (page 313)

11.2.9.13 obj_type=KC_TACCLASS

This operation relates to a UTM application TAC class.

In the identification area you must pass the number of the TAC class (field *kc_name2* of the union *kc_id_area*). In the data area you must pass the data structure *kc_tacclass_str* with the new values of the properties.

- *Possible modification*

  You can increase or decrease the number of processes which may simultaneously process jobs for transaction codes of the TAC class. To do this, you can:

  – Specify the number of processes in absolute terms (*tasks*), i.e.:

    you specify the number of processes which may simultaneously perform jobs for this TAC class. If the number is specified in absolute terms, the number of processes is independent of the currently set total number of processes in which the application program is running. This applies provided that the current total number of process in the application is no less than the number of processes set for the TAC class. If this is case, the number of processes is reduced accordingly.

  – Specify the number of processes in relative terms (*tasks_free*), i.e.:

    you specify the number of processes which must remain free to process jobs for transaction codes of other TAC classes. If the number is stated in relative terms, the number of processes for this TAC class is determined by the currently set total number of application processes. If the total number of processes is reduced, then the maximum number of processes which process jobs for the TAC class is also reduced implicitly. Similarly, if the total number is increased, the number of processes for this TAC class is also increased implicitly.

  The number of processes of a TAC class can only be modified, if the application was generated without priority control, i.e. if the KDCDEF generation does not contain a TAC-PRIORITIES statement.

For this modification, you must specify the following in the structure *kc_tacclass_str*:

Field name	Meaning
tasks	Specify in *tasks* the maximum number of processes which may simultaneously perform jobs for transaction codes of the TAC class. A relative statement previously made by *tasks_free* for this TAC class is deactivated. Minimum value of *tasks*: For dialog TAC classes (TAC classes 1-8), *tasks* must be ≥ ´1´, as dialog services would otherwise be locked and users would have to wait at the terminal until the processes were released again. For asynchronous TAC classes (classes 9-16) *tasks* may be ≥ ´0´. Maximum value: see table on page 353. If the value specified for *tasks* is greater than the total number of processes for the application, then UTM automatically reduces the value to this number.
tasks_free	Specify the following in *tasks_free*: – for dialog TAC classes: the minimum number of processes which are to be kept free to process jobs for other TAC classes. If the number of processes in *tasks_free* becomes greater than the total number of processes available to the application program, then one process nevertheless remains available to this TAC class to process its transaction codes. – for asynchronous TAC classes: the minimum number of processes which are to be kept free to process transaction codes of other asynchronous TAC classes. If the number of processes in *tasks_free* becomes greater than the total number of processes which may simultaneously be used for asynchronous processing, then no further jobs are performed on transaction codes of this TAC class. Minimum value: ´0´ Maximum value: see table on page 353.

tasks and *tasks_free* must not be specified together in a single KC_MODIFY_OBJECT call.

The permitted maximum value for *tasks* and *tasks_free* is determined by the following factors:

– whether or not program units with blocking calls (*pgwt*=´Y´) can run in the TAC class.

– by the values for TASKS, TASKS-IN-PGWT and ASYNTASKS generated statically in the KDCDEF control statement MAX.

The following table contains the maximum permitted values for *tasks* and *tasks_free*. If you specify greater values, the KC_MODIFY_OBJECT call is rejected.

TAC class	Content of pgwt	Permitted maximum value for tasks	Permitted maximum value for tasks_free
1 - 8 (dialog TACs)	´N´	TASKS *)	TASKS - 1 *)
	´Y´	TASKS-IN-PGWT *)	TASKS - 1 *)
9 - 16 (asynchronous TACs)	´N´	ASYNTASKS *)	ASYNTASKS *)
	´Y´	the smaller of the values: ASYNTASKS, TASKS-IN-PGWT*)	ASYNTASKS *)

*) As statically generated in the KDCDEF control statement MAX

Period of validity / transaction management: Type A (page 313)

C KDCTCL (page 792)

- Reset the statistical values “Average wait time of the jobs in the job queues” and “Number of wait situations”. These two values can only be reset together.

  The values can be reset either for the TAC class specified in the Id area or for all the TAC classes:

  – If the values are to be reset for all TAC classes then binary zero must be specified in the Id area. In this case, *tasks* and *tasks_free* must not be modified.

  – If only a specific TAC class is to be modified then *avg_wait_time_msec* and *nr_waits* can be specified together with *tasks* and *tasks_free*.

  Specify the following in the *kc_tacclass_str* data structure:

Field name	Meaning
avg_wait_time_msec[10]	Contains the average wait time of the jobs in the job queues assigned to the transaction codes of this TAC class. The unit of the *avg_wait_time_msec* value is milliseconds. If there is no process available for the TAC class, UTM accepts jobs for the TAC class (with free processes that “cannot” process jobs to this TAC class) and stores them temporarily in the KDCFILE. This is always the case when there are jobs pending for TAC classes of a higher priority (in the case of priority control) or when the number of processes is limited and the maximum permitted number of processes is already processing transaction codes of the TAC class (see *tasks*, *tasks_free*). The time between the acceptance of a job and the start of its processing is the wait time displayed here. You can reset this value to ´0´.

Field name	Meaning
nr_waits[10]	Number of wait situations taken into account for the calculation of the value *avg_wait_time_msec*. You can reset this value to ’0’.
nr_calls[10]	Number of proram unit runs for this TAC class.

Period of validity / transaction management: type GIR (page 313)

11.2.9.14 obj_type=KC_TPOOL

This operation relates to an LTERM pool for the UTM application.

In the identification area you must pass the name of the LTERM pool (LTERM prefix). For this the field *kc_name8* of the union *kc_id_area* is available.

In the data area you must pass the data structure *kc_tpool_str* with the new values of the properties.

Possible modification

- You can increase or decrease the number of clients which may be connected concurrently via this LTERM pool, i.e. you specify how many LTERM partners of the LTERM pool are to be released or disabled. One client can connect to the application via each enabled LTERM partner in the LTERM pool. The number of LTERM partners included in the LTERM pool, i.e. the maximum number of LTERM partners which can be permitted for this LTERM pool, is defined during KDCDEF generation. Specify the following in the data structure *kc_tpool_str*:

Field name	Meaning
state=´N´ state_number=...	Of the total number of LTERM partners in this LTERM pool (see *kc_tpool_str.max_number* on page 564), the number specified in *state_number* is to be disabled. The number of permitted LTERM partners for this LTERM pool is consequently: *max_number* - *state_number*. If the entire LTERM pool is to be disabled, you must specify the value of *max_number* in *state_number*. If you want to release all the LTERM partners in the LTERM pool, specify *state_number*= ´0´. Minimum value for *state_number*: ´0´ Maximum value for *state_number*: the maximum number returned in *kc_tpool_str.max_number*
state=´Y´ state_number=...	Of the total number of LTERM partners, only the number specified in *state_number* is to be permitted. If all the LTERM partners in the LTERM pools are to be permitted, you must specify the generated maximum value (*kc_tpool_str.max_number* on page 564) in *state_number*. You can disable the entire LTERM pool if you specify *state_number*=´0´. Minimum value for *state_number*: ´0´ Maximum value for *state_number*: the maximum number returned in *kc_tpool_str.max_number*

The fields *state* and *state_number* must always be specified together.

If the number in *state_number* exceeds the generated maximum number of LTERM partners, UTM automatically resets the value of *state_number* to this maximum number.

Period of validity / transaction management: type GP (page 313)

Disabling LTERM partners in the LTERM pool has the following effect:

– A connection setup request from a client via this LTERM pool is rejected by UTM as soon as the permitted number of clients is reached which are connected to the application via this LTERM pool (all permitted LTERM partners are occupied).

– If, at the time at which the call is processed by UTM, the number of live connections to this LTERM pool exceeds the number of permitted LTERM partners for the LTERM pool, all existing connections are initially maintained.

  The lock only comes into effect for new connection setup requests.

  If terminal users sign off with KDCOFF BUT, they can sign on again with KDCSIGN, even if at that time more clients than permitted are connected to the application through the LTERM pool. This is possible because the connection remains in this case.

KDCPOOL (page 762)

- You can change the maximum period for which UTM waits for an entry from the client after the end of a transaction or after sign-on. If the time is exceeded, the connection to the client is cleared down. Specify the following in the *kc_tpool_str* data structure:

Field name	Meaning
idletime[5]	In *idletime* you specify the maximum time in seconds that openUTM waits for an entry from the client outside a transaction (i.e. after the end of a transaction or after sign-on). When *idletime*=0 is specified, openUTM waits for an unlimited period. Maximum value: ´32767´ Minimum value: ´60´, In the case of values that are smaller than 60 but not equal to 0 then the value 60 is used.

Period of validity / transaction management: type GP (page 313)

The modification of the timer takes effect at the next end of transaction, but not before the end of the program unit run (PEND) in which the call is processed.

11.2.9.15 obj_type=KC_USER

This operation relates to a UTM application user ID and its queue.

In the identification area you must specify the name of the user ID (field *kc_name8* of the union *kc_id_area*). In the data area you must pass the data structure *kc_user_str* with the new values of the properties.

Possible modifications

- Lock or release a user ID.

  Neither users nor clients can then sign on to the application under a locked user ID. User IDs with administration privileges cannot be locked.

Field name	Meaning
state=´N´	The user ID is to be disabled. If the user is signed on to the application at the time at which the user ID is disabled, the user is not disconnected. The lock does not take effect until the user or client next attempts to sign on to the application under this user ID. Read and write accesses to the queue of a locked user ID are possible.
state=´Y´	The user ID is to be released, i.e. there is a lock in existence which is to be cancelled.

  Period of validity / transaction management: type GPD (page 313)

- Change the key set assigned to the user ID. Specify the following in the *kc_user_str* data structure:

Field name	Meaning
kset[8]	In *kset* you specify the name of an existing key set that sets the access rights of the user ID in the application. The name of a key set can be up to 8 characters long. The user or client program can only access a service protected by means of a lock code or an access list if: – the key set of the user ID and – the key set of the LTERM partner by means of which the terminal user or the client program connects to the application contain a key/access code that corresponds either to the lock code of the service or to at least one key of the access list of the service. If you want to cancel the assignment that has applied up to now, enter blanks.

  Period of validity / transaction management: type GPD (page 313)

- Change or delete the password for a user ID.

  When changing a password, you must take account of the level of complexity and minimum password length defined when the user ID was created. You can ascertain the level of complexity and minimum length using KC_GET_OBJECT (object type KC_USER). UTM reports the settings in the fields *protect_pw_compl* and *protect_pw_lth* of the data structure *kc_user_str*. The levels of complexity and the criteria which must be fulfilled by a password of a certain level of complexity are described on page 582.

  You can only delete passwords if:

  – the minimum password length defined when the user ID was created (*protect_pw_lth*) is equal to ´0´ and

  – no particular level of complexity is defined for the user (*protect_pw_compl*=´0´).

  If a password with a limited period of validity has been defined for a user ID (*protect_pw_time*≠´0´, page 582), you cannot use the old password as the new password when changing the password.

  In applications generated with SIGNON GRACE=Y, you can choose one of the following options when changing the password (*protect_pw_time_left*):

  – the generated period of validity is to apply to the new password (from the time the change is implemented) or

  – the password is to become invalid immediately and must be changed immediately the next time the user signs on.

  If a password with a limited period of validity is deleted, no period of validity applies. If a new password is issued subsequently, the period of validity again takes effect.

When changing a password, you must specify both the new password and the password type. Specify the following in the data structure *kc_user_str*:

	Field name	Meaning
B B B B B B B X/W X/W X/W X/W	password	Specify the new password for this user ID in the *password* field. You must also specify in the *password_type* field how UTM is to interpret the value specified in *password*. In the *protect_pw_time_left* field you can prevent a password with a limited period of validity from becoming invalid immediately in applications generated with SIGNON GRACE=Y. If the password is invalid, it is necessary to assign a new password at sign-on. The password can be up to 8 characters long. The union *kc_pw* is available for passing the password (see page 224). In UTM applications running on BS2000 systems you can specify the password either as a character string or as a sequence of hexadecimal characters. In a hexadecimal password, each half byte is represented as a character. If you specify a password which consists of less than 8 characters, *password* must be padded to the right with blanks (*password_type*= ´C´), or with the hexadecimal value for blanks (*password_type*=´X´). In UTM running on Unix, Linux or Windows systems, you must always pass the password as a character string. If you specify a password consisting of less than 8 characters, the field must be padded to the right with blanks. In order to delete a password, specify only blanks in *password* or specify 'N' in *password_type*
B B B B X/W X/W	password_type	If you want to change the password, you must specify in *password_type* how the password specified in *password* is to be interpreted. The following specified values are possible: – UTM applications on BS2000 systems: ´C´, if the new password is a character string or ´X´, if the new password is specified as a sequence of hexadecimal characters – UTM applications on Unix, Linux and Windows systems: ´C´, since the password must be passed as a character string. If you want to delete the password of a user ID, pass ´N´ in *password_type*. In this case, nothing further need be specified in *password*

Period of validity / transaction management: type GPD (page 313)

Field name	Meaning
protect_pw_time_left	This only applies to applications generated with SIGNON GRACE=Y and for user IDs whose passwords are generated with a limited period of validity. In *protect_pw_time_left*, you can specify whether the generated period of validity is to apply to the new password: If you enter *protect_pw_time_left*='-1' (right or left-justified) the generated period of validity applies to the new password (from the time it was implemented). *protect_pw_time_left*='-1' only has effect together with *password* and *password_type*. *protect_pw_time_left*='-1' without a password is ignored. If you make no entries for *protect_pw_time_left* the password immediately becomes invalid, because the period of validity is expired. The user must change the password at the next sign-on. A value other than '-1' is rejected.

- You can change write, read and delete authorization for a USER queue. Specify the following in the *kc_user_str* data structure:

Field name	Meaning
q_read_acl[8]	In *q_read_acl* you specify the name of an existing key set by means of which the queue is protected against other users who want to access the queue to read and delete messages. You can remove the protection by specifying blanks. In this case, all users can read and delete messages from this queue.
q_write_acl[8]	In *q_write_acl* you specify the name of an existing key set by means of which the queue is protected against other users who want write access to it. You can remove the protection by specifying blanks. In this case, all users can write messages to this queue.

Another user (≠*us_name*) can have read (delete) or write access to the USER queue when both the key set of the user's user ID and the key set of the LTERM partner by means of which the user is signed on contain at least one key code of the *q_read_acl* or *q_write_acl* key set.

Period of validity / transaction management: type GPD (page 313)

- Assign a new start format to the user ID.

  You can assign a specific start format to each user ID. This start format is automatically output after each successful sign-on if no service is currently open for this user ID. In order to modify the start format, you must always specify both the format name and the format attribute.

  The precondition for assigning a start format is that a formatting system has been generated (KDCDEF statement FORMSYS). If the start format is a #format, a sign-on service must also be generated.

Field name	Meaning
format_attr	Format identifier of the new start format: ´A´ for the format attribute ATTR. The format name at the KDCS program interface is +*format_name*. ´N´ for the format attribute NOATTR. The format name at the KDCS program interface is ∗*format_name*. ´E´ for the format attribute EXTEND. The format name at the KDCS program interface is #*format_name*. The meanings of the format attributes are described on page 580.
format_name[7]	Name of the start format. The name can be up to 7 characters long and may contain only alphanumeric characters.

  If you want to delete the start format of a user ID, you must specify blanks in *format_attr* and *format_name*.

  Period of validity / transaction management: type GPD (page 313)

- Enable or disable the BCAM trace for this user ID.

  To allow USER-specific enabling:

  The BCAM trace must not be generally enabled for all connections, i.e. the trace is either completely disabled or only explicitly enabled for certain selected LTERM and LPAP partners or USERs.

  Specify the following in the data structure *kc_user_str*:

Field name	Meaning
bcam_trace='Y'	The BCAM trace is explicitly enabled for this USER. This is only possible – if the BCAM trace is disabled for all connections (see *kc_diag_and_account_par_str*) or – if the BCAM trace has already been enabled for individual USERs.
bcam_trace='N'	The BCAM trace is disabled for this USER.

  Period of validity / transaction management: type GIR (page 313)

  Some modifications can also be performed using KDCUSER (page 797).

11.2.9.16 obj_type=KC_CLUSTER_CURR_PAR

In UTM cluster applications, it is necessary to reset the statistics values of the cluster page pool.

You must enter the data structure *kc_cluster_curr_par_str* via the data area.

Possible modifications

The following table indicates the values you are able to reset.

Field name	Meaning
max_cpgpool_size=´0´	Maximum utilization of the cluster page pool. The counter is reset to 0.
avg_cpgpool_size=´0´	Average utilization of the cluster page pool. The counter is reset to 0.

If you reset one of the two values then the other value is also implicitly reset.

Period of validity / transaction management: type GID (page 313)

Unless explicitly reset, the values continue to apply after the complete cluster application has terminated and are not reset until the size of the cluster page pool is increased and the UTM cluster files are generated using KDCDE.

11.2.9.17 obj_type=KC_CLUSTER_PAR

You want to modify the circular monitoring settings for the node applications in a UTM cluster application and/or the settings for node application access to the cluster configuration file and the administration journal of the UTM cluster application.

To do this, you must enter the new property values in the the data structure *kc_cluster_par_str* via the data area.

Possible modification

The following table indicates the settings that you are able to modify.

Field name	Meaning
check_alive_timer _sec	In a UTM cluster application, every node application is monitored by another node application (circular monitoring), i.e. each node application monitors the availability of another node application and is itself monitored by a node application. To do this, the monitoring node application sends messages to the monitored node application at defined intervals (*check_alive_timer_sec*). If the monitored application is available, it acknowledges the message. *check_alive_timer_sec* specifies the interval in seconds at which monitoring messages are sent to the monitored node application. openUTM also uses this timer in order to access the cluster configuration file and the administration journal periodically in order to check for possible updates. Minimum value ‘30’’ Maximum value ‘3600’’
communication _retry communication _reply_timer_sec	*communication_retry* specifies how often a node application repeats an attempt to send a monitoring message if the monitored node application does not respond within the defined time. *communication_reply_timer_sec* specifies the maximum time in seconds that a node application waits for a response after sending a monitoring message. If the monitored node application does not respond in the defined time, then it is assumed to have failed (abnormal end of application) and the command sequence defined in *failure_cmd* is executed (e.g. a restart). If a value greater than zero is set for *communication_retry*, then the target node application is only assumed to have failed if, additionally, no response to the monitoring message is received after the final retry.
	communication_retry: Minimum value: ‘0’’ Maximum value: ‘10’’ *communication_reply_timer_sec:* Minimum value: ‘1’ Maximum value: ’60’

Field name	Meaning
restart_timer_sec	Maximum time in seconds that a node application requires for a warm start after a failure. If a value of 0 is specified, no timer is set for monitoring the restart of a failed node application. Minimum value: 0, i.e. restart of the application is not monitored. Maximum value: 3600
file_lock_timer_sec file_lock_retry	*file_lock_timer_sec* is the maximum time in seconds that a node application waits for a lock to be assigned for accessing the cluster configuration file or the cluster administration journal. *file_lock_retry* specifies how often a node application repeats the request for a lock on the cluster configuration file or the cluster administration journal if the lock was not assigned in the time specified in *file_lock_timer_sec*. Note: Do not choose too small a value since a timeout when accessing the cluster configuration file can lead to the abnormal termination of the application. *file_lock_timer_sec*: Minimum value: ‘10’ Maximum value: ‘60' *file_lock_retry*: Minimum value: ‘1' Maximum value: ‘10'
deadlock_prevention=´N´	UTM does not perform any additional verifications for the GSSB, TLS and ULS data areas in order to prevent deadlocks. If a deadlock occurs in one of these data areas then it is resolved via a timeout.
deadlock_prevention=´Y´	UTM performs additional verifications for the GSSB, TLS and ULS data areas in order to prevent deadlocks. In productive operation it is advisable to set this parameter to ´Y´ only if timeouts occur frequently when accessing these data areas.

Period of validity / transaction management: type GID (page 313)

11.2.9.18 obj_type=KC_CURR_PAR

The counters for application-specific statistical values are to be reset. For this you must assign the data structure *kc_curr_par_str* to the data area.

Furthermore, you can enable or disable data compression, see page 368.

Possible modifications

All the counters listed below can be set in one call. In order to reset the counters you must pass the value ´0´ to UTM in the relevant field, unless there is a note to the contrary. You can reset the following counters or statistical values:

Field name	Specification
term_input_msgs=´0´	Number of messages which the application received from clients or partner applications since the last reset. The counter is reset to 0.
term_output_msgs=´0´	Number of messages which the application sent to clients, printers or partner applications since the last reset. The counter is reset to 0.
max_dial_ta_per_100sec=´0´	Maximum number of dialog transactions carried out within the space of 100 seconds. The counter is reset to 0 (page 604).
max_asyn_ta_per_100sec=´0´	Maximum number of asynchronous transactions carried out within the space of 100 seconds. The counter is reset to to 0 (page 604).
max_dial_step_per_100sec=´0´	Maximum number of dialog steps carried out within the space of 100 seconds. The counter is reset to to 0 (page 604).
max_pool_size=´0´	Maximum utilization of the page pool in percent since the last reset. The counter is reset to 0. If this value is reset then the value of *avg_pool_size* is also implicitly reset to 0.
avg_pool_size=´0´	Average utilization of the page pool in percent since the last reset of the counter. The counter is reset to 0. If this value is reset then the value of *max_pool_size* is also implicitly set to 0.
cache_hit_rate=´0´	Hit rate for pages in the cache memory since the counter was last reset (in percent). The counter is reset to 0. If this value is reset then the values *cache_wait_buffer*, *nr_cache_rqs* and *nr_cache_searches* are also impicitly reset to 0.

Field name	Specification
cache_wait_buffer=´0´	Percentage of buffer requests in the cache, that led to a wait time. The counter is reset to 0. If this value is reset then the values *cache_hit_rate* , *nr_cache_rqs* and *nr_cache_searches* are also impicitly reset to 0.
abterm_services=´0´	Number of abnormally terminated services since the last reset. The counter is reset to 0.
deadlocks=´0´	Number of known and resolved deadlocks of UTM resources since the last reset. The counter is reset to 0.
periodic_writes=´0´	Number of periodic writes since the last reset (periodic write = backup of all relevant administration data in the UTM application). The counter is reset to 0.
pages_pwrite=´0´	Number of UTM pages saved on average in a periodic write. The counter is reset to 0.
logfile_writes=´0´	Number of requests to write log records to the user log file ((USLOG). The counter is reset to 0.
maximum_jr=´0´	In distributed processing only: Maximum number of remote job receiver services addressed in the local application at the same time in relation to the generated value MAXJR (see *kc_utmd_par_str* on page 661). This is a percent value The counter is reset to the value of *curr_jr* (page 606).
max_load=´0´	*max_load* specifies as a percentage the maximum load of the UTM application registered since the start of the application or the last reset. The value is reset to the value in *curr_load* (see page 607).
max_wait_resources=´0´	*max_wait_resources* specifies the maximum conflict rate for user data locks over the application run. The value is specified as an amount per thousand. The counter is reset to 0. If this value is reset then the values *max_wait_system_resources*, *nr_res_rqs_for_max* and *nr_sys_res_rqs_for_max* are also impicitly reset to 0.

Field name	Specification
max_wait_system_resources=´0´	*max_wait_system_resources* specifies the maximum conflict rate for system resource locks (system locks) across the application run. The value is specified as an amount per thousand. The counter is reset to 0. If this value is reset then the values *max_wait_resources*, *nr_res_rqs_for_max* and *nr_sys_res_rqs_for_max* are also implicitly reset to 0.
nr_cache_rqs=´0´	Number of buffer requests taken into account to calculate the value *cache_wait_buffer*. The counter is reset to 0. If this value is reset then the values *cache_hit_rate*, *cache_wait_buffer* and *nr_cache_searches* are also implicitly reset to 0.
nr_cache_searches=´0´	Number of search operations for UTM pages in the cache taken into account to calculate the value *cache_hit_rate*. The counter is reset to 0. If this value is reset then the values *cache_hit_rate*, *cache_wait_buffer* and *nr_cache_rqs* are also implicitly reset to 0.
nr_res_rqs_for_max=´0´	Number of requests for transaction resources in the 100 second period during which the maximum conflict rate *max_wait_resources* was reached. The counter is reset to 0. If this value is reset then the values *max_wait_resources*, *max_wait_system_resources* and *nr_sys_res_rqs_for_max* are also implicitly reset to 0.
nr_sys_res_rqs_for_max=´0´	Number of requests for system resources in the 100 second period during which the maximum conflict rate *max_wait_system_resources* was reached. The counter is reset to 0. If this value is reset then the values *max_wait_resources*, *max_wait_system_resources* and *nr_res_rqs_for_max* are also implicitly reset to 0.
avg_saved_pgs_by_compr=´0´	Average value for the UTM pages saved per data compression. The counter is reset to 0.

Period of validity / transaction management: type GIR (page 313)

If you wish to reset the statistical values listed above yourself, you should set MAX STATISTICS-MSG =NONE in KDCDEF generation. This stops UTM resetting the counters to 0 at hourly intervals and creating the statistics message K081.

Enabling/Disabling data compression

Field name	Specification
data_compression=´Y´	Data compression is enabled. For this purpose data compression must be permitted by means of UTM generation, see openUTM manual “Generating Applications”, MAX DATA-COMPRESSION=
data_compression=´N´	Data compression is disabled.

Period of validity / transaction management: type GIR (page 313)

11.2.9.19 obj_type=KC_DIAG_AND_ACCOUNT_PAR

Diagnostic functions are to be activated or deactivated. You must pass the data structure *kc_diag_and_account_par_str* in the data area.

Possible modifications

- Activate or deactivate the ADMI trace function. The ADMI trace function logs all calls of the KDCADMI program interface.

Field name	Meaning
admi_trace=´Y´	The ADMI trace function is enabled.
admi_trace=´N´	The ADMI trace function is disabled. All ADMI trace files are closed and can be analyzed. For more information, see also openUTM manual ”Messages, Debugging and Diagnostics”.

Period of validity / transaction management: Type IR (page 313)

It is also possible to enable the trace via the start parameters when the application is started, see openUTM manual “Using openUTM Applications”. The names of the trace files are also described there.

- Activate or deactivate BCAM trace for all connections to the application, i.e. for all:
  - LTERM partners, LPAP partners
  - USER
  - MUX connections

B

BCAM trace records all connection-related events.

Field name	Meaning
bcam_trace=´Y´	The BTRACE function is activated for all connections. When the BTRACE function is activated, each application process creates its own trace file in which it records connection-related events.
bcam_trace=´N´	The BTRACE function is deactivated for all connections, even if it had previously only been activated for specific LTERM, LPAP, MUX or USER. If the BTRACE function is deactivated (for all LTERM, LPAP, MUX partners and USERs), the trace files are closed and can be evaluated subsequently. Trace file content and evaluation are described in the openUTM manual ”Messages, Debugging and Diagnostics”.

You can also activate or deactivate the BCAM trace LTERM-, LPAP-, MUX or USER-specifically. Use the object types KC_LTERM (page 332), KC_LPAP (page 325), KC_MUX (page 336) or KC_USER (page 357) for this purpose.

Period of validity / transaction management: type GIR (page 313)

The BCAM trace can be activated by means of start parameters when the application is started.

- Control the CPI-C trace function. The CPI-C trace function logs calls at the X/Open interface CPI-C.

Field name	Meaning
cpic_trace=´T´	The CPI-C trace function is enabled with the level TRACE. The content of the input and output parameters is output for each CPI-C function call. Only the first 16 bytes are output from the data buffers. The return codes of the KDCS calls to which the CPI-C calls are mapped are output.
cpic_trace=´B´	The CPI-C trace function is enabled with the level BUFFER. This trace level includes the TRACE level. However, the data buffers are logged in their full length.
cpic_trace=´D´	The CPI-C trace function is enabled with the level DUMP. This trace level includes the TRACE level and also writes diagnostic information to the trace file.
cpic_trace=´A´	The CPI-C trace function is enabled with the level ALL. This trace level includes the levels BUFFER, DUMP and TRACE.
cpic_trace=´N´	The CPI-C trace function is disabled (OFF). All CPI-C trace files are closed and can be analyzed. For more information, see also openUTM manual “Creating Applications with X/Open Interfaces”.

Period of validity / transaction management: Type IR (page 313)

It is also possible to enable the CPI-C trace via the start parameters when the application is started, see openUTM manual “Using openUTM Applications”. The names of the trace files are also described there.

- Activate or deactivate OSI trace functions for all application OSI connections.

  The OSI trace functions record all events occurring during distributed processing through OSI TP. The events recorded are restricted to certain record types, i.e. to events relating to certain components.

It is not possible to deactivate logging for individual record types. If the trace is to be deactivated for individual record types, it must first be completely deactivated (*osi_trace*=´N´) and then reactivated for those record types that are still to be logged (appropriate specified values in *osi_trace_records*).

Field name	Meaning
osi_trace=´Y´	The OSI trace function is activated for all record types. When the OSI trace function is activated, each application process creates its own trace file.
osi_trace=´N´	The OSI trace is deactivated for all record types. All OSI trace files are closed and can be evaluated. See also openUTM manual ”Messages, Debugging and Diagnostics”.
osi_trace_records[5]	Activate the OSI trace function for certain record types. Nothing further need be specified in the *osi_trace* field to activate the OSI trace. Each field element of *osi_trace_records* represents a record type: 1st field, record type “SPI“ 2nd field, record type “INT“ 3rd field, record type “OSS“ 4th field, record type “SERV“ 5th field, record type “PROT“ The meaning of the record types is summarized on page 613. To activate trace functions for certain record types, specify ‘Y’ in the appropriate field elements. The call activates logging for the specified record types in addition to any log files that may already exist.

Period of validity / transaction management: type GIR (page 313)

Tracing can be activated by means of start parameters when the application is started.

- Control the TX trace function. The TX trace function logs calls at the X/Open interface TX.

Field name	Meaning
tx_trace=´E´	The TX trace function is enabled with the level ERROR. Only errors are logged.
tx_trace=´I´	The TX trace function is enabled with the level INTERFACE. The level INTERFACE includes the level ERROR, and all TX calls are also logged.
tx_trace=´F´	The TX trace function is enabled with the level FULL. The FULL level includes the INTERFACE level. All KDCS calls to which the TX calls are mapped are also logged.

Field name	Meaning
tx_trace=´D´	The TX trace function is enabled with the level DEBUG. The level DEBUG includes the level FULL, and diagnostic information is also logged.
tx_trace=´N´	The TX trace function is disabled. All TX trace files are closed and can be analyzed. For more information, see also openUTM manual “Creating Applications with X/Open Interfaces”.

Period of validity / transaction management: Type IR (page 313)

It is also possible to enable the TX trace via the start parameters when the application is started, see openUTM manual “Using openUTM Applications”. The names of the trace files are also described there.

- Control the XATMI trace function. The XATMI trace function logs calls at the X/Open interface XATMI.

Field name	Meaning
xatmi_trace=´E´	The XATMI trace function is enabled with the level ERROR. Only errors are logged.
xatmi_trace=´I´	The XATMI trace function is enabled with the level INTERFACE. The level INTERFACE includes the level ERROR, and all XATMI calls are also logged.
xatmi_trace=´F´	The XATMI trace function is enabled with the level FULL. The FULL level includes the INTERFACE level. All KDCS calls to which the XATMI calls are mapped are also logged.
xatmi_trace=´D´	The XATMI trace function is enabled with the level DEBUG. The level DEBUG includes the level FULL, and diagnostic information is also logged.
xatmi_trace=´N´	The XATMI trace function is disabled. All XATMI trace files are closed and can be analyzed. For more information, see also openUTM manual “Creating Applications with X/Open Interfaces”.

Period of validity / transaction management: Type IR (page 313)

It is also possible to enable the XATMI trace via the start parameters when the application is started, see openUTM manual “Using openUTM Applications”. The names of the trace files are also described there.

- Activate and deactivate application test mode.

  Test mode should only be activated to generate diagnostic documents. Internal UTM plausibility check routines also run in test mode and internal TRACE data is recorded.

Field name	Meaning
testmode=´Y´	Test mode is activated (ON).
testmode=´N´	Test mode is deactivated (OFF).

  Period of validity / transaction management: type GIR (page 313)

  Test mode can be activated by means of start parameters when the application is started.

- You can create a diagnostic dump for defined messages/events.

  You can define an event for which, on its occurrence, UTM generates a diagnostic dump which contains an event-dependent ID. The prerequisite for this is that test mode must be activated (*testmode*=´Y´). Test mode can be activated and the event defined in a KC_MODIFY_OBJECT call. You can also define the event when test mode is not activated. However, the diagnostic dump is only written on the occurrence of the event when test mode is activated.

  You can specify the following events:

  – the output of a specific K or P message, possibly depending on the inserts in the message

  – the occurrence of a specific KDCS return code (KCRCCC or KCRCDC) in a program unit run

  – the occurrence of a specific SIGN status when a user signs on

  The events are specified in *kc_diag_and_accout_par_str* in the data structure *kc_dump_event_str*, which contains the data structure *kc_insert_str* in addition to the fields *event_type* and *event*.

  Any message inserts which further restrict generation of the dump are defined in *kc_insert_str*. You can specify up to three inserts. A dump is only generated if all the criteria for the message inserts specified in kc_insert_str apply.

Data structure kc_dump_event_str

Field name	Meaning
event_type[4]	Type of event for which a UTM dump is to be generated:
	MSG K or P message RCDC Incompatible return code RCCC Compatible return code SIGN SIGNON status code NONE Explicit deactivation of an individual event
event[4]	Message number, KDCS return code (CC or DC) or SIGNON status code, depending on the *event_type*

Data structure kc_insert_str

Field name	Meaning

value[64]	*value* can be specified as follows, depending on *value_type*:
	value_type=N numeric, integers between 0 and 2^{31}-1
	value_type=C alphanumeric, maximum of 32 characters
	value_type=X hexadecimal, maximum of 64 characters
	UTM represents the string in a union of the type *kc_value*:
	`union kc_value`
	`{ char x[64];`
	`  char c[32];`
	`};`

value_type	*value_type* specifies how the contents of the field *value* are to be interpreted: N numeric C alphanumeric X hexadecimal
comp[2]	Specifies whether the system is to test for equality or inequality. The possible values are EQ (equality) or NE (inequality)

In the case of messages K023, K043, K061 or K062, UTM creates a UTM dump only once, namely when the message next occurs. The message dump function is then automatically deactivated.

In the case of all other UTM message numbers, a UTM dump is created each time the specified event occurs. This is done until the event is explicitly reset.

In the case of KDCS return codes or SIGNON status codes, the function is automatically deactivated after the message dump has been generated.

Period of validity / transaction management: type GIR (page 313)

KDCDIAG (page 681)

- You can activate and deactivate the accounting and calculation phase of UTM Accounting.

  See also the openUTM manual “Generating Applications” and the openUTM manual “Using openUTM Applications” for information on accounting in UTM.

BBBB

Field name	Meaning
account=´Y´	Activate the accounting phase. UTM Accounting is always deactivated after a BS2000 accounting failure, even if BS2000 accounting is still available. UTM accounting must then be reactivated with *account* =´Y´.
account=´N´	Deactivate the accounting phase (OFF).
calc=´Y´	Activate the calculation phase in UTM accounting (ON).
calc=´N´	Deactivate the calculation phase of UTM accounting (OFF).

  Period of validity / transaction management: type GIR (page 313)

  After the application is started, the value set in ACCOUNT ACC= during KDCDEF generation applies.

- Activate or deactivate the event monitor KDCMON

  See the openUTM manual “Using openUTM Applications” in relation to event monitor KDCMON and the UTM tools for evaluating the measured values (KDCEVAL).

Field	Meaning
kdcmon=´Y´	Activate KDCMON (ON)
kdcmon=´N´	Deactivate KDCMON (OFF)

  Period of validity / transaction management: type IR (page 313)

KDCDIAG (page 681) / KDCAPPL (page 666)

- Switch over the log files from the UTM application.

  It is possible to switch over the log files for the application (SYSOUT and SYSLST or stderr and stdout) during live operation. This allows you to avoid a disk bottleneck and permits evaluation and archiving of the log files while the application is running.

Field name	Meaning
sysprot_switch='Y'	The log files are switched over.

Period of applicability / transaction management: type GA (page 313)

KDCAPPL (page 666)

B

- Enable or disable Stxit logging

B

Field name	Meaning
stxit_log='Y'	Enables Stxit logging.
stxit_log='N'	Disables Stxit logging.

B
B

If STXIT logging is enabled, multiple K099 messages are output to SYSOUT when an STXIT event occurs.

B

Period of applicability / transaction management: type IR (page 313)

B

KDCDIAG (page 681)

- Output debug information for the database connection.

  You can specify the extent to which calls to the XA interface will be logged and the destination for such logging.

Field name	Meaning
xa_debug='Y'	Enables XA-DEBUG (ON). Calls to the XA interface are logged.
xa_debug='A'	Extended XA-DEBUG (ALL). Specific data areas are output in addition to the calls to the XA interface.
xa_debug='N'	Disables XA-DEBUG (OFF).
xa_debug_out='S'	Output to SYSOUT/stderr.
xa_debug_out='F'	Output to a file.

  If you use only the field xa_debug without providing a value for xa_debug_out, any value you specified in the start parameter when starting the UTM application will be used (see openUTM Manual “Using openUTM Applications”). Otherwise, the log is written to SYSOUT/stderr.

  Period of applicability / transaction management: type IR (page 313)

  KDCDIAG (page 681)

11.2.9.20 obj_type=KC_MAX_PAR

Application parameters and maximum values for the application are to be modified. You must assign the data structure *kc_max_par_str* in the data area.

Possible modifications

All the modifications described below can proceed in a single call.

- You can modify application maximum values, which were defined in the MAX statement during KDCDEF generation. These modifications may affect application performance (see also "Performance check" on page 48).

  The following table shows which maximum values can be modified and the fields of the data structure *kc_max_par_str* to which you must pass the new maximum values.

Field name	Meaning
bretrynr[5]	Specify in *bretrynr* how often UTM is to attempt to pass a message to the transport system (BCAM) if BCAM cannot immediately accept the message. The selected value of *bretrynr* should not be too high because the process attempting to the pass the message to BCAM is blocked for the duration of the attempts. For asynchronous messages to a dialog partner type of the *ptype*=´APPLI´ (TS application), *bretrynr* is not relevant (see *bretrynr* on page 628) Minimum value: ´1´ Maximum value: ´32767´
cachesize_paging[3]	Specify in *cachesize_paging* the percentage of the cache which is to be written to the KDCFILE in the event of a bottleneck so that the cache memory can be used for other data. UTM replaces at least 8 UTM pages out to cache in a single paging, even if the value of *cachesize_paging* is smaller. Minimum value: ´0´, i.e. 8 UTM pages are swapped out to cache Maximum value: ´100´ (%) Cache size is defined in the MAX statement during KDCDEF generation and can be ascertained, for example, by using KC_GET_OBJECT for *obj_type*=KC_MAX_PAR (*cache_size_pages*).

Period of validity / transaction management: type GIR (page 313)

Field name	Meaning
conn_users[10]	By using *conn_users* you can prevent the application from being overloaded by too many active users. To do this, specify in *conn_users* the maximum number of users or clients that can currently be signed on to the UTM application. The following situation applies in applications generated with user IDs: – If the number specified for *conn_users* is greater than the number of generated users, *conn_users* has no effect. – User IDs which have been generated with administration privileges can still sign on to the UTM application after the maximum number of concurrent user IDs has been reached. The following situation applies in applications which are generated without user IDs: – The number of dialog partners which can concurrently be connected to the UTM application is restricted by *conn_users*. – If the number specified for *conn_users* is greater than the number of generated dialog LTERM partners, *conn_users* has no effect. Dialog LTERM partners are all those LTERM partners entered with *usage_type*= ´D´, LTERM partners of the LTERM pool and the LTERM partners created internally by UTM for multiplex connections. If the number of simultaneously active users is not to be restricted or if a restriction is to be cancelled, specify *conn_users*= ´0´. Minimum value: ´0´ (i.e. no restriction) Maximum value: ´500000´ On Unix, Linux and Windows systems, the maximum value may not exceed the value generated in the generation parameter MAX ... CONN-USERS.

X/W
X/W
X/W

Period of validity / transaction management: type IR (page 313)

You can define a new destination for the results of the KDCADM administration commands which were called by KDCADM through asynchronous TACs.

Field name	Meaning
destadm[8]	Specify in *destadm* the new recipient for the results of KDCADM administration calls which have been processed asynchronously (asynchronous KDCADM transaction codes). This overwrites the old value of *destadm*. You can specify the following for *destadm*: – the name of an LTERM partner – an asynchronous transaction code or – a TAC queue If you specify blanks for *destadm* no recipient is defined any longer. The results of the asynchronous KDCADM transaction code then are lost.

Period of validity / transaction management: type GPD (page 313)

- You can change the number of failed attempts which UTM allows before UTM triggers the silent alarm.

Field name	Meaning
signon_fail	Specify in *signon_fail* the number of unsuccessful sign-on attempts (security violations) from a client following in immediate succession after which a “silent alarm” (K094-UTM message) is triggered. Minimum value: ´1´ Maximum value: ´100´

Period of validity / transaction management: type GIR (page 313)

- You can activate or deactivate the supply of data to openSM2:

Field name	Meaning
sm2=´Y´	UTM is to supply data to openSM2 for the purpose of monitoring performance data. The supply of data can only be activated if it was not excluded at a general level during KDCDEF generation (MAX statement operand SM2).
sm2=´N´	The supply of data to openSM2 is to be deactivated.

Period of validity / transaction management: type GIR (page 313)

Some of the modifications can also be performed with the administration command KDCAPPL (page 666).

11.2.9.21 obj_type=KC_TASKS_PAR

The values relating to the number of application processes can to be modified, i.e. the total number of processes, maximum number of processes for processing asynchronous jobs and for processing program units with blocking calls and the number of processes reserved for UTM-internal jobs and dialog jobs that do not belong to a TAC class.

You must assign the data structure *kc_tasks_par_str* in the data area.

Possible modifications

All the modifications described below can be made in a single call.

Field name	Meaning
mod_max_tasks[3];	Change the total number of processes running. In this field you specify the maximum number of processes that are running for the application. *mod_max_tasks* is a target value for the current number of processes. The number of actually active processes that currently process jobs of the application is stored in the *curr_tasks* field (see *kc_tasks_par_str* as of page 652). This can differ from *mod_max_tasks* for a short period at the startup or termination of a process. Maximum value: the maximum value (*tasks*) defined in MAX at KDCDEF generation Minimum value: ´1´
mod_max_asyntasks[3]	Modify the maximum number of processes that can process asynchronous jobs simultaneously. Specify in *mod_max_asyntasks* the maximum number of processes that can simultaneously be used for asynchronous processing. The number specified here serves as a upper limit value. The actual maximum number of processes that can be used concurrently for asynchronous processing (see *kc_tasks_par_str* as of page 652, *curr_max_asyntasks* parameter) may be lower than the value specified in *mod_max_asyntasks*, because the actual number is limited by the number of processes of the application that are currently running (*curr_tasks*). Minimum value: ´0´ Maximum value: the maximum value defined in MAX at KDCDEF generation (*asyntasks).*

Field name	Meaning
mod_max_tasks_in_pgwt[3]	Modifies the maximum number of processes which may simultaneously process jobs for program units in which blocking calls are permitted. Specify in *mod_max_tasks_in_pgwt* the maximum number of processes in which program units that have blocking calls can run simultaneously. The number specified here serves as a upper limit value. The actual maximum number of processes processing program units with blocking calls simultaneously (see *kc_tasks_par_str* as of page 652, *curr_max_tasks_in_pgwt* parameter) may be lower than the value specified in *mod_max_tasks_in_pgwt* because the actual number must be at least 1 below the number of currently running processes of the application (*curr_tasks*). *mod_max_tasks_in_pgwt*=´0´ is rejected if the application contains transaction codes or TAC classes with *pgwt*=´Y´. Minimum value: ´0´ Maximum value: the maximum value defined in MAX during KDCDEF generation (*tasks_in_pgwt)*.
mod_free_dial_tasks[3]	This value can only be modified if a TAC-PRIORITIES statement was issued during KDCDEF generation. In *mod_free_dial_tasks*, you enter the number of processes in the application reserved for UTM-internal jobs and for dialog jobs that do not belong to a specific dialog TAC class. This portion of the total percentage is then not available for processing jobs to dialog TAC classes. If *mod_free_dial_tasks* ≥ *mod_max_tasks* after the process figures have been modified, an application process may still process jobs to dialog TAC classes. Minimum value: ´0´ Maximum value: value in *tasks* -1

Period of validity / transaction management: type A (page 313)

KDCAPPL (page 666)

11.2.9.22 obj_type=KC_TIMER_PAR

Application timer settings are to be modified. You must enter the data structure *kc_timer_par_str* in the data area.

Possible modifications

The following table shows which timers can be modified. You can modify as many of these timers as you wish in a single call.

Field name	Meaning
conrtime_min[5]	Specify here the time in minutes after which UTM is to attempt to re-establish a lost connection to a printer or a TS application. The precondition is that the connection must previously have been established automatically by UTM (*kc_pterm_str.auto_connect*=´Y´ or *kc_lterm_str.plev* > 0; see also page 657). At *conrtime_min*=´0´ UTM makes no attempt to re-establish a lost connection. Maximum value: ´32767´ Minimum value: ´0´
pgwttime_sec[5]	The maximum time in seconds which a program unit is to wait for the arrival of messages after a blocking function call (e.g. PGWT). During this waiting period, one process remains exclusively reserved by this program unit. Maximum value: ´32767´ Minimum value: ´60´
reswait_ta_sec[5]	The maximum time in seconds which a program unit is to wait for a device currently being used by another transaction. See also page 658 in relation to *reswait_ta_sec*. *reswait_ta_sec*=´0´ means that the program unit does not wait. A program unit run wishing to access a reserved device immediately receives an appropriate return code. Maximum value: ´32767´ Minimum value: ´0´

	Field name	Meaning
	reswait_pr_sec[5]	The maximum time in seconds which UTM is to wait for a device currently being used by another process. If this time is exceeded, the application terminates with a UTM error message. It should be noted that the value of *reswait_pr_sec* must be as long as the longest (real time) processing time for the following cases: – In TS applications that are not SOCKET applications (clients with PTYPE=APPLI) the devices are locked for the duration of a processing stage, including a VORGANG exit at the beginning and/or end of the service. – At the end of the service, the devices are reserved for as long as the VORGANG exit program is running. Minimum value: ´300´, Maximum value: ´32767´ If you specify a value of < 300, the call is rejected.
	termwait_in_ta_sec[5]	The maximum time in seconds in a multi-step transaction (i.e. in the PEND KP program) which may elapse between an output to a dialog partner and the subsequent dialog response. If the time *termwait_in_ta_sec* is exceeded, the transaction is rolled back. The devices reserved by the transaction are released. The connection to the partner is shut down. Maximum value: ´32767´ Minimum value: ´60´
B B B B B	logackwait_sec[5]	The maximum time in seconds which UTM is to wait for a logical print confirmation from the printer or a transport confirmation for an asynchronous message to another application (created using the KDCS call FPUT). If the confirmation does not arrive after this time, e.g. due to a printer being out of paper, UTM shuts down the logical connection to the device. Minimum value: ´10´ Maximum value: ´32767´
B B		

Field name	Meaning
The following timers are relevant only in the context of UTM applications with distributed processing via LU 6.1 or OSI TP.	
conctime1_sec[5]	The time in seconds for monitoring the setup of a session (LU6.1) or association (OSI TP). If the session or association is not established within the specified time, UTM shuts down the transport connection to the partner application. *conctime1_sec*=´0´ means: – for LU6.1 connections: session setup is not monitored (UTM will wait indefinitely). – for OSI TP connections: UTM waits up to 60 seconds for an association to be set up. Minimum value: ´0´ Maximum value: ´32767´
conctime2_sec[5]	The maximum waiting time in seconds for a confirmation from the recipient when transferring an asynchronous message. Once the time *conctime2_sec* has expired, UTM shuts down the transport connection. The asynchronous job is not lost, but remains in the local message queue. *conctime2_sec* = ´0´ means that monitoring is not performed. Minimum value: ´0´ Maximum value: ´32767´
ptctime_sec[5]	This timer is relevant only in the context of distributed processing via LU6.1 connections. *ptctime_sec* defines the maximum time in seconds which a local job-receiving service will wait in the PTC state (prepare to commit, transaction status P) for a confirmation from the job-submitting service. When the time expires, the connection to the job submitter is shut down, the transaction in the job-receiving service is rolled back and the service terminated. This may possibly result in a mismatch. If KDCSHUT WARN or GRACE has already been issued for the application and the value of *ptc_time_sec* is not 0, then the waiting time is chosen independently of *ptc_time_sec* in such a way that the transaction is rolled back before the application is terminated in order to avoid abnormal termination of the application with ENDPET if possible. *ptctime_sec* = ´0´ means that UTM waits indefinitely for a confirmation. Minimum value: ´0´ Maximum value: ´32767´

Period of validity / transaction management: type GIR (page 313)

The modifications do not take effect on timers which are already running; they only apply to timers started after the modification.

Some of the modifications can also be performed with the administration command KDCAPPL (page 666).

11.2.9.23 Return codes

In addition to the return codes listed in section “Return codes” on page 174, the following codes can also occur. Some of these return codes may occur independently of the specified object type; others occur only for certain object types.

Type-independent return codes:

Main code = KC_MC_DATA_INVALID Information is missing from the data structure in the data area or a field contains an invalid value. **Subcodes:**
KC_SC_DATA_MISSING Data is missing from the data structure. Possible causes: – The field to be modified was not specified. – Several fields must be specified together for the requested modification, and one of these values is missing (e.g. *obj_type*=KC_TPOOL: *state* and *state_number*).
KC_SC_INVALID_MOD A field in the data structure which can be modified was completed with an invalid value.
KC_SC_NOT_NULL A field in the data structure which cannot be modified was not completed with binary zero.

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the present time. **Subcode:**
KC_SC_INVDEF_RUNNING An inverse KDCDEF is currently running and configuration data cannot be changed during the run.

Main code = KC_MC_NOT_EXISTENT No object of the type specified in *obj_type* exists under the name or name triplet passed in the identification area. **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_DELETED The specified object has been deleted. Its properties cannot be modified. **Subcode:**
KC_SC_NO_INFO

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_NOT_GEN No explicitly generated object of the object type specified in *obj_type* exists. Implicitly generated objects might, however, exist, e.g. user IDs for clients with *ptype*=´APPLI´.
KC_SC_JCTL_RT_CODE_NOT_OK Only in UTM cluster applications: An internal UTM error occurred during the global modification of an object. Please contact system support.
KC_SC_NO_CLUSTER_APPLI This action is only possible in a UTM cluster application.
KC_SC_NO_GLOB_CHANG_POSSIBLE No global administration changes are possible since the generations of the node applications are not consistent at present.
KC_SC_NOT_ALLOWED_IN_CLUSTER The administration action is not permitted in a UTM cluster application.

Main code = KC_MC_RECBUF_FULL The buffer with recovery information is full (see KDCDEF control statement MAX, operand RECBUF). **Subcode:**
KC_SC_NO_INFO

Return codes for obj_type=KC_CLUSTER_NODE:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_CCFG_NO_CLUSTER_APPLI The specified application is not a UTM cluster application
KC_SC_CCFG_FILE_NOT_OPEN Internal UTM error. Please contact system support.
KC_SC_CCFG_RT_CODE_NOT_OK Modification was not performed. Possible cause, e.g. timer expired.
KC_SC_CCFG_FILE_LOCK_ERROR Cluster configuration file is locked.
KC_SC_CCFG_FILE_READ_ERROR Error reading the cluster configuration file.

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_CCFG_FILE_WRITE_ERROR Error writing the cluster configuration file.
KC_SC_CCFG_INVALID_BUFFER_LTH Internal UTM error. Please contact system support.
KC_SC_CCFG_INVALID_NODE_INDEX Internal UTM error. Please contact system support.
KC_SC_CCFG_INVALID_NODE_STATE Invalid node application status. Note: You may not make any modifications for a running node application.
KC_SC_CCFG_INVAL_FILEBASE_NAME Base name of UTM cluster invalid.
KC_SC_CCFG_INVALID_HOSTNAME The host name is invalid.

Return codes for obj_type=KC_KSET:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED It is not permissible to modify the KDCAPLKS or MASTER key set.

Return codes for obj_type=KC_LOAD_MODULE (program replacement):

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the present time. **Subcode:**
KC_SC_CHANGE_RUNNING A program replacement is running.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_CHANGEABLE The load module / shared object / DLL specified in the identification area is not interchangeable. Possible reasons include, for example: – the load module has the load mode STATIC. – the load module contains TCB entries.
KC_SC_SAME_VERSION *load_mode* ≠´U´ (not STARTUP): The currently loaded version of the load module was specified in *version*.

Return codes for obj_type=KC_LPAP:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CONNECTED *state* = ´N´: There is a connection to the partner application. The partner application thus cannot be disabled. Before the partner application is disabled, all connections to it must be shut down.
KC_SC_NOT_ALLOWED Possible causes: – you have attempted to establish a connection to a disabled partner application (*state* = ´N´) with *connect_mode* = ´Y´, or – you have set *state* = ´N´ together with *connect_mode* = ´Y´, or – you have specified *connect_mode* and *quiet_connect* together, or – the value specified in *bcam_trace* is not permissible.
KC_SC_NOT_EXISTENT The specified object does not exist.

Return codes for obj_type=KC_LSES:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED Possible causes: – The combination of the specified modifications is not permitted, i.e. both *connect_mode* and *quiet_connect* were set. – There is no connection to the partner application and it is not possible to establish one because the LPAP partner of the partner application is disabled. The LPAP partner must first be enabled in a separate transaction.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_CON The connection specified by (*con*, *pronam*, *bcamappl*) is invalid. It does not exist or is intended for another partner application (LPAP partner).
KC_SC_CONNECTED A connection to be established was specified in (*con*, *pronam*, *bcamappl*). However, the session already has another connection.

Maincode = KC_MC_NOT_EXISTENT The specified object does not exist. **Subcode:**
KC_SC_NO_INFO No LU6.1 connection was created or generated.

Return codes for obj_type=KC_LTAC:

There are no type-specific return codes for KC_LTAC.

Return codes for obj_type=KC_LTERM:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_POOL_LTERM The LTERM partner specified in the identification area belongs to an LTERM pool. The requested modification is not permissible for this LTERM partner.
KC_SC_NO_PTERM *connect_mode* = ´Y´ was set: UTM cannot establish a connection because no client/printer is currently assigned to the LTERM partner or the associated client/printer is disabled.
KC_SC_NOT_ALLOWED Possible causes: – an attempt was made to define a start format for an LTERM partner with *usage_type*=´O´. – *format_attr*=´E´ (#format) was specified, but no sign-on service is defined. – an inadmissible value was specified in *bcam_trace*. – The replacement of two master LTERMs was rejected because one of the LTERMs is not a master LTERM or the same master was specified for both. The replacement of two master LTERMs is not permitted in a UTM cluster application.
KC_SC_NO_FORMAT_ALLOWED Values specified in *format_name* and *format_attr* (modifying the start format) are not permitted as no formatting system has been generated for the application.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_INVALID_ALIAS The primary LTERM is itself an alias LTERM.
KC_SC_INVALID_ALIAS_CTERM The primary LTERM is a CTERM.
KC_SC_INVALID_ALIAS_BUNDLE The primary LTERM is a slave LTERM in an LTERM bundle.
KC_SC_ALIAS_STATE_ILL The primary LTERM has been generated with RESTART=NO or QAMSG=NO.

B **Return codes for obj_type=KC_MUX:**

Main code = KC_MC_REJECTED The call was rejected by UTM. Subcodes:
KC_SC_CONNECTED *state*=´N´: There is a connection to the multiplex connection. It therefore cannot be disabled. *connect_mode* = ´Y´: There is already a connection to the multiplex connection.
KC_SC_NOT_ALLOWED You have tried to establish a connection to a disabled multiplex connection, or the value specified in *bcam_trace* is not permitted.

Return codes for obj_type=KC_OSI_CON:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_CONNECTED There is a connection to the partner application. It is only possible to switch to a replacement connection if no active association to the partner application currently exists.

Return codes for obj_type=KC_OSI_LPAP:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CONNECTED Specified value *state* = ´N´: a connection to the partner application exists. The OSI-LPAP partner of the partner application therefore cannot be disabled. All connections to the partner application must be shut down before the disable operation.
KC_SC_NOT_ALLOWED Possible causes: – you have attempted to establish a connection (*connect_number*>0) to a disabled partner application (OSI-LPAP partner) or to a partner application for which no connection is set to active (see *kc_osi_con_str* field *active*) – you have set *state* = ´N´ together with *connect_number*, or – you have set *state* = ´N´ together with *quiet_connect*, or – you have set *quiet_connect* together with *connect_number*.

Return codes for obj_type=KC_PTERM:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED Possible causes: – an attempt was made to establish a connection to a disabled client/printer, or – *connect_mode* = ´R´ is not permitted for the client specified in the identification area, or – the fields *lterm* and *connect_mode* were specified together. – *state* = ´N´ and *auto_connect* = ´Y´ were specified together.
KC_SC_POOL_PTERM The requested modification is not permitted for clients connected via an LTERM pool.
KC_SC_UPIC_PTERM The requested modification is not permitted for clients with *ptype*= ´UPIC-R´ or ´UPIC-L´ (on Unix, Linux and Windows systems).
X/W X/W KC_SC_TTY_PTERM The requested modification is not permitted for a terminal (*ptype*=´TTY´).
B B B B B KC_SC_MUX_DIS_PENDING The specified client is connected to the application via a multiplex connection and the session is in the state DISCONNECT PENDING. An attempt was made either to establish or shut down the session (*connect_mode*=´Y´ or ´N') or to release the session explicitly while the timer was still running (*connect_mode*=´R´).

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_LTERM_NOT_EXISTENT The client/printer assignment to the LTERM partner cannot be modified as the LTERM partner specified in *lterm* does not exist.
KC_SC_LTERM_DEL The client/printer assignment to the LTERM partner cannot be modified as the LTERM partner specified in *lterm* has been deleted.
KC_SC_LTERM_NOT_ALLOWED The client/printer assignment to the LTERM partner cannot be modified. Possible causes: – The LTERM partner specified in *lterm* belongs to an LTERM pool. – The specified LTERM partner has been configured for connection to a client with *ptype*=´UPIC-...´ and cannot be assigned to any other client. – KDCMSGLT was specified in *lterm*. KDCMSGLT is generated internally by UTM for the event service MSGTAC. It cannot be assigned to any client/printer.
KC_SC_CONNECTED The client/printer assignment to the LTERM partner cannot be modified. Possible causes: – The client/printer which is to be assigned to the LTERM partner is currently connected to the application. – A client which is connected to the application is currently assigned to the LTERM partner. The old assignment of the LTERM partner cannot be cancelled as one of the two clients is entered as a dialog partner (*usage_type*=´D´).
KC_SC_OUT_PTERM_DIAL_LTERM The name of an output medium (*usage_type*=´O´) was stated in the identification area, but the LTERM partner specified in *lterm* is configured as a dialog partner. An output medium cannot be assigned to a dialog LTERM partner.
KC_SC_DIAL_PTERM_TO_BUNDLE The new client/printer assignment to the LTERM partner cannot be created. The name of a dialog partner (*usage_type*=´D´) was passed in the identification area, but the LTERM partner specified in *lterm* belongs to a printer pool.
KC_SC_PTYPE_APPLI The new client/printer assignment to the LTERM partner cannot be created. The name of a client having *ptype*=´APPLI´ or 'SOCKET' was specified in the identification area. The LTERM partner specified in *lterm* is not suitable for this client because no user ID has been generated for the LTERM partner.
KC_SC_PTERM_WITHOUT_CID The new client/printer assignment to the LTERM partner cannot be created. The specified LTERM partner is assigned to a printer control LTERM, but no printer ID (CID) has been defined for the specified printer.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CID_AMBIGUOUS The new client/printer assignment to the LTERM partner cannot be created. The specified LTERM partner is assigned to a printer control LTERM, but the printer ID defined for the specified printer is not unambiguous at the level of the printer control LTERM.
KC_SC_NO_LTERM *connect_mode* = ´Y´ is not permitted: no LTERM partner is assigned to the specified client/printer, so no connection can be established.
KC_SC_INVALID_PROTOCOL_USAGE PTYPE and protocol cannot be combined.
KC_SC_BUNDLE_NOT_ALLOWED It is not possible to make the new assignment between the client and the LTERM partner because the LTERM partner belongs to an LTERM bundle.
KC_SC_GROUP_NOT_ALLOWED It is not possible to make the new assignment between the client and the LTERM partner because the LTERM partner belongs to an LTERM group.
KC_SC_NOT_ALLOWED_IN_CLUSTER This function is not permitted in a UTM cluster application, e.g. KDCSWTCH or replacement of two bundle masters

Return codes for obj_type=KC_TAC:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_NOT_ALLOWED Possible causes: – An attempt was made to modify *state* and to reset statistics values at the same time. – It is possible that an attempt was made to modify the *lock_code* and *access_list* parameters. It is not permitted to modify *access_list* if *lock_code* is generated. – It is not permitted to modify *access_list* in the case of the TACs KDCBADTC, KDCMSGTC and KDCSGNTC. – An attempt was made to disable KDCTAC. – A TAC generated with the NEXT property should be disabled with *state*='N'. This is not permissible. Disabling it has no effect. – In the case of a TAC that is not of the type 'Q', an attempt was made to modify 'q_read_acl' or 'q_write_acl'. – An attempt was made to set *dead_letter_q* = 'Y' for an interactive or asynchronous TAC with CALL=NEXT or for a KDCDLETQ or KDCMSGTC TAC.
KC_SC_INVALID_READ_ACL The key set specified in *q_read_acl* does not exist.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_INVALID_WRITE_ACL The key set specified in *q_write_acl* does not exist.
KC_SC_INVALID_ACL The key set specified in *access_list* does not exist.
KC_SC_READ_ACL_DEL The key set was deleted.
KC_SC_WRITE_ACL_DEL The key set was deleted.

Return codes for obj_type=KC_TACCLASS:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED – An invalid number of processes was specified in *tasks* or *tasks_free*. – Both *tasks* and *tasks_free* were specified.
KC_SC_NOT_CHANGEABLE *tasks* and *tasks_free* cannot be modified because the application was generated with priority control (TAC-PRIORITIES).

Return codes for obj_type = KC_TPOOL:

There are no type-specific return codes for KC_TPOOL.

Return codes for obj_type=KC_USER:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_TOO_SIMPLE The requested password change was not performed as the new password is not of the complexity level (*protect_pw_compl*) defined for the user ID.
KC_SC_OLD_PW The requested password change was not performed as the old password was specified in *password* and a limited period of validity is defined in the user ID for the password (*protect_pw_time*≠´0´). The old password cannot be specified as the new password for this user ID.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_ALLOWED The requested modification was not performed. Possible causes: – *state*=´N´: you have attempted to disable a user ID that has administration privileges (permit=´A´ or ´B´). – you have attempted to modify a user ID which is assigned to a client having *ptype*=´APPLI´, 'SOCKET' or ´UPIC-...´. – you have attempted to modify the user ID KDCMSGUS which UTM has generated internally for the event exit MSGTAC. – you have specified *format_attr*=´E´ (#format), but no sign-on service has been defined. – It is only permitted to enable or disable the BCAM trace if the BTRACE module is set to SELECT mode.
KC_SC_NO_FORMAT_ALLOWED It is not permitted to specify information in *format_name* and *format_attr* (modifying the start format), as no formatting system has been generated for the application.
KC_SC_INVALID_READ_ACL The key set specified in *q_read_acl* does not exist.
KC_SC_INVALID_WRITE_ACL The key set specified in *q_write_acl* does not exist.
KC_SC_READ_ACL_DEL The referenced key set was deleted.
KC_SC_WRITE_ACL_DEL The specified key set was deleted.
KC_SC_KSET_DEL The referenced key set was deleted.
KC_SC_KSET_NOT_EXISTENT The specified key set does not exist.
BB KC_SC_INVALID_CERTIFICATE Error on sign-on with certificate.
BB KC_SC_INVALID_PRINCIPAL Error on sign-on with principal.

Return codes for obj_type=KC_CLUSTER_PAR:

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CCFG_NO_CLUSTER_APPLI The application is not a UTM cluster application.

Maincode = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_CCFG_RT_CODE_NOT_OK Modification was not performed. Internal UTM error. Please contact system support.
KC_SC_CCFG_FILE_LOCK_ERROR Cluster configuration file is locked.
KC_SC_CCFG_FILE_WRITE_ERROR Error writing the cluster configuration file.
KC_SC_CCFG_FILE_READ_ERROR Error reading the cluster configuration file.
KC_SC_INVALID_BUFFER_LTH Internal UTM error. Please contact system support.
KC_SC_CCFG_FILE_NOT_OPEN Internal UTM error. Please contact system support.

Return codes for obj_type=KC_DIAG_AND_ACCOUNT:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_AVAILABLE The event monitor KDCMON cannot be activated. It is not available.
KC_SC_NOT_GEN The OSI trace is to be activated although no objects have been generated for distributed processing through OSI TP.
KC_SC_SYSPROT_SWITCH_RUNNING A log file is currently in the process of being switched over to the next log file. It is therefore not possible to execute a new switchover command.
KC_SC_TRCFILE_HANDLING_RUNNING Trace files are currently being opened or closed, with the result that it is not possible to modify the trace settings at present.

Return codes for obj_type=KC_MAX_PAR:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
BB KC_SC_NOT_GEN Data supply to openSM2 was not generated, i.e. it cannot be activated or deactivated.
BB KC_SC_NOT_AVAILABLE openSM2 is currently unavailable.
KC_SC_NOT_ALLOWED An invalid destination was specified when modifying *destadm* (recipient of results from KDCADM asynchronous TACs). Possible causes: – an LTERM partner which has been disabled or deleted was specified in *destadm*. – a transaction code which has been disabled or deleted was specified in *destadm*. – a dialog TAC was specified in *destadm*, but only an asynchronous TAC or an LTERM partner may be specified as the recipient. – an LTERM partner was specified in *destadm* to which a client of the type UPIC_... is assigned.
KC_SC_NOT_EXISTENT Invalid information in *destadm*. The specified name belongs neither to an LTERM partner nor to a transaction code.

Return codes for obj_type=KC_TASKS_PAR:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_NOT_ALLOWED – The number of processes specified in *mod_max_tasks*, *mod_max_asyntasks* or *mod_max_tasks_in_pgwt* is greater than the value generated in the KDCDEF statement MAX. – *mod_max_tasks_in_pgwt*=´0´ is not allowed, since the application allows blocking call, i.e. transaction codes or TAC classes with *pgwt*=´Y´ were generated.

Return codes for obj_type=KC_TIMER_PAR:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_NO_UTMD An attempt was made to set a timer for distributed processing through LU6.1 or OSI TP, although no objects have been generated for distributed processing.

11.2.10 KC_ONLINE_IMPORT - Import application data online

In a UTM cluster application, following the normal termination of a node application, another running node application can import messages to LTERMs, (OSI) LPAPs, asynchronous TACs, TAC queues and open asynchronous services from the terminated node application provided that its KDCFILE comes from the same generation run. The imported data is deleted in the terminated node application. Prior to import, a check is performed to determine whether an online import is running. If it is, the new import is rejected. Online import is only possible in UTM-S applications. Open asynchronous services are not imported if the service contains database transactions with SESAM/SQL.

Execution / period of validity / transaction management / clusters

KC_ONLINE_IMPORT initiates the online import of the application data, i.e. an online import job is generated. When control returns to the program unit, the online import has not yet been performed. Online imports are not subject to transaction management. It cannot be rolled back by a subsequent RSET call in the same transaction. Online import is performed by a process in the application.

When the job has been processed, UTM issues a message informing you of the success or failure of the online import. If the import was successful but it was not possible to import all the data due to a temporary resource bottleneck, another online import can be run to import the outstanding data into another node application or, once the bottleneck has been cleared, into the same node application.

This function is only permitted in cluster operation. The online import operation is performed in the node application in which the call is made.

Parameter settings

Parameter area

	Field name	Contents
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_ONLINE_IMPORT
1.	subopcode1	KC_ALL
	id_lth	0
	select_lth	0
2.	data_lth	Length of the data structure

Identification area

—

Selection area

—

Data area

3.	Data structure

Data returned by UTM

Parameter area

	Field name	Content
4.	retcode	Return codes

1. With *subopcode1*=KC_ALL, you specify that all messages, i.e. messages to (OSI) LPAPs, asynchronous TACs, TAC queues and open asynchronous services are to be imported.
2. In *data_lth*, you enter the length of the data structure in the data area.

3. Specify the data structure *kc_online_import_str* in the data area.

   In *kc_online_import_str*, specify the number of the node from which the application data is to be imported.

   The data structure *kc_online_import_str* is defined as follows.

Data structure kc_online_import_str
`char import_node[4];`

   The field in the data structure has the following meaning:

   import_node
   Number of the node from which the application data is to be imported.

4. openUTM indicates the return code from the call in the *retcode* field. Alongside the return codes listed in section “Return codes” on page 174, the following return codes may also occur:

Maincode = KC_MC_REJECTED The call was rejected by openUTM. **Subcode:**
KC_ONLINE_IMPORT_RUNNING An attempt has been made to start an online import while an online import is already running.
KC_SC_CCFG_INVALID_NODE_INDEX The number of the node application from which the application data is to be imported is invalid. The number is either the number of the local node application or a number that does not belong to the UTM cluster application.
KC_SC_CCFG_INVALID_NODE_STATE The node application from which the application data is to be imported has a status that is not valid for online imports. An invalid status means that the node application – has either never been started, or – has been terminated abnormally, or – is not running

Maincode = KC_MC_NOT_EXISTENT The number of the node application from which the import is to be performed lies outside of the valid range of values from 1 to 32. **Subcode:**
KC_SC_NO_INFO

11.2.11 KC_PTC_TA - Roll back transaction in PTC state

KC_PTC_TA rolls back a transaction that is in the state PTC (prepare to commit).

The transaction's identification data consists of a triad of elements: the service index, service number and transaction number. You can obtain this data by first issuing a KC_GET_OBJECT call with operation code KC_PTC.

Execution / period of validity / transaction management / cluster

This call rolls back the local element of a distributed transaction.

The distributed transaction itself cannot be be rolled back using the administration capabilities. Only the local element of such a transaction can be rolled back. This type of administrative rollback is a heuristic decision concerning the result of the transaction and may in certain cases lead to inconsistencies in the distributed data stock if the distributed transaction is committed by the Commit Coordinator.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_PTC_TA
1.	subopcode1	KC_ROLLBACK
2.	id_lth	25
	select_lth	0
	data_lth	0

Identification area

3.	Triad with the transaction's identification data

Selection area

—

Data area

—

Data returned by UTM

Parameter area

4.

Field name	Content
retcode	Return codes

1. With *subopcode1*=KC_ROLLBACK, you specify that the transaction is to be rolled back.
2. You specify the length of the data structure *kc_ptc_id_str* in the *id_lth* field.
3. In the identification area, you specify the data structure *kc_pct_id_str.*

   kc_ptc_id_str must be filled with the values returned by the call KC_GET_OBJECT with operation code KC_PTC in the structure *ptc_ident*. *ptc_ident* is present in the data structure *kc_ptc_str*, see page 527.

   The data structure *kc_ptc_id_str* is defined as follows.

Data structure kc_ptc_id_str
`char vg_indx[10];`
`char vg_nr[10];`
`char ta_nr_in_vg[5];`

   vg_indx is the index of the service, *vg_nr* the number of the service and *ta_nr_in_vg* the number of the transaction in the service.
4. openUTM returns the return code for the call in the *retcode* field. Alongside the return codes listed in section “Return codes” on page 174, the following return codes may also occur

Maincode = KC_MC_REJECTED The call was rejected by openUTM. **Subcode:**
KC_SC_NO_MORE_PTC The transaction is no longer in the PTC state.
KC_SC_END_TA_ALREADY_INITIATED The termination of the transaction has already been initiated. There may be the following reasons for this: – The partner of the distributed transaction that determines the result of the transaction (Commit Coordinator) has initiated the termination of the transaction – The termination of the transaction has been initiated by the administration functions.

Maincode = KC_MC_REJECTED The call was rejected by openUTM. **Subcode:**
KC_SC_PARTNER_CONNECTED The connection has been established to the partner of the distributed transaction that determines the result of the transaction (Commit Coordinator). This initiates termination of the transaction.

11.2.12 KC_SEND_MESSAGE - Send message (BS2000 systems)

B Using KC_SEND_MESSAGE, you can send a message to one or more or all active terminals of a UTM application on a BS2000 system. The message text may be up to 74 characters in length and it is passed to UTM in the data area. UTM then sends the message as UTM message K023 with the specified message as an insert. By default, the message is output in the system line on the terminal. However, the message destination of message K023 can also be changed. If the message destination PARTNER is selected for the UTM message K023 (see the openUTM manual ”Messages, Debugging and Diagnostics”), you can also send the message to one or more or all connected TS applications. The message only goes to dialog partners (LTERM with USAGE=D).

B Using KC_SEND_MESSAGE, you can:

- B send a message to all terminals currently connected to the application. This also applies to terminals connected to the application via an LTERM pool.
- B send a message to all TS applications connected to the UTM application, provided the message destination PARTNER is generated for K023.
- B send a message to a certain terminal user or, provided the message destination PARTNER is generated, to a specific TS application. In this case, you must specify in the identification area the name of the LTERM partner via which the terminal is connected to the application. The precondition for delivery of the message is that the terminal must be connected to the application at the time the KC_SEND_MESSAGE call is issued.

B If you want to send a message to a certain user, you can ascertain the LTERM partner through which the user is signed on to the application in the following manner:

B First, using KC_GET_OBJECT, request information about the user ID under which the user has signed on to the application (object type KC_USER).

B UTM then returns the properties of the user ID in the data structure *kc_user_str*. If, at the time of the request, the user is connected to the application, the field *lterm_curr* contains the name of the LTERM partner through which the user is signed on. This is the name which you pass in the identification area when sending the message with KC_SEND_MESSAGE.

Execution / transaction management / cluster

A KC_SEND_MESSAGE call is not subject to transaction management. It cannot be rolled back by an RSET in the same transaction.

If you do not specify a recipient in the identification area and the parameter area to *number*=0, UTM identifies all currently active LTERM partners entered with *usage_type*=´D´ and sends them the message. The message will already have been sent when control is returned to the program unit.

If you specify the name of an LTERM partner in the identification area and set the parameter area to *number*=1, successful processing of the KC_SEND_MESSAGE call means that the message has been sent to this LTERM partner. If the LTERM partner cannot currently be reached, UTM returns an appropriate return code.

The following applies to UTM cluster applications:
The call is effective locally in the node, i.e. the transmission of a message to one, several or all active UTM terminals of a UTM application on a BS2000 system is executed only in the node application at which the call is performed.

KDCSEND (page 774)

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identification area**	**selection area**	**data area**
Send message to all active LTERM partners	*obj_number*: 0	——	——	Message
Send message to one LTERM partner	*obj_number*: 1	Name of LTERM partner	——	Message

[1] The operation code KC_SEND_MESSAGE must always be specified in the parameter area.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_SEND_MESSAGE
1.	obj_number	1 / 0
2.	id_lth	Length of object name / 0
	select_lth	0
3.	data_lth	Length of message

Identification area

4.	Object name / —

Selection area

—

Data area

5.	Message

KDCADMI call

```
KDCADMI (&parameter_area, &identification_area, NULL, &data_area) or
KDCADMI (&parameter_area, NULL, NULL, &data_area)
```

Data returned by UTM

Parameter area

	Field name	Content
6.	retcode	Return codes

1. Specify in *obj_number* whether the message is to be sent to all currently active LTERM partners or only to a specific LTERM partner.

   – *obj_number*=0 means:
   The message is to be sent to all active LTERM partners. The null pointer must be passed as the address of the identification area.

   – *obj_number*=1 means:
   The message is to be sent to only one LTERM partner. The name of the LTERM partner must be passed in the identification area.

2. The length of the identification area must be specified in *id_lth*, i.e.:

   – for *obj_number*=0 you must specify *id_lth*=0.

   – for *obj_number*=1 you must specify in *id_lth* the length of the object name which is passed in the identification area.

3. Length of the message to be sent. You must pass the message in the data area. The following must apply: $1 \leq data\_lth \leq 74$.

4. How you have to complete the identification area depends on the value set for *obj_number*.

   – for *obj_number*= 0 you must pass the null pointer in the KC_SEND_MESSAGE call.

   – for *obj_number*= 1 you must specify in the identification area the union *kc_id_area* with the name of the LTERM partner (field *kc_name8*), to which the message is to be sent.

5. The message which UTM is to send is to be passed in the data area. The message must be no more than 74 characters in length.

6. UTM writes the return codes for the call to the *retcode* field.
In addition to the return codes listed in section “Return codes” on page 174, the following codes can also occur.

Main code = KC_MC_REJECTED The call was rejected by UTM. Subcodes:
KC_SC_NOT_EXISTENT The name specified in the identification area is unknown, no LTERM partner with this name exists.
KC_SC_NOT_ALLOWED The operation is not allowed for the LTERM partner specified in the identification area or for the client assigned to this LTERM partner. Possible reasons for rejection are: – there is currently no connection to the client; the LTERM partner is not active – no client is currently assigned to the LTERM partner – the specified LTERM partner is not a dialog partner, i.e. it has been configured with *usage_type*=´O´ – the client assigned to the specified LTERM partner has been deleted from the configuration.
KC_SC_DELETED The specified LTERM partner no longer exists, it has been deleted from the application configuration.

11.2.13 KC_SHUTDOWN - Terminate the application run

Using KC_SHUTDOWN you can terminate the current application run.

In UTM cluster applications, you can specify whether the application run is to be terminated at all nodes or only at the node at which the call is issued.

The following options are open to you:

- You can terminate the application run normally. UTM terminates the application run as soon as all running dialog steps have terminated (KC_NORMAL).
- You can schedule the application to terminate after a specified period (KC_WARN).
- You can terminate the application once all the UTM-D dialogs have been terminated and all the UTM-D connections have been disconnected and at the latest, however, after a specified period (KC_GRACEFUL).
- You can abort the application, i.e. immediately terminate (KC_KILL).

See also the openUTM manual “Using openUTM Applications” for more information on terminating a UTM application.

Please note the following when aborting the application:

Aborting the application (KC_KILL) cannot be handled as an asynchronous service: it is only permitted as a dialog. A call containing *subopcode1*=KC_KILL in an asynchronous service is rejected by UTM.

Please note the following when shutting down applications involving distributed processing:

You should preferably terminate applications with distributed processing with KC_GRACEFUL, alternatively with KC_WARN. When doing this, you should specify a time that is greater than the maximum period that a distributed transaction remains in the state PTC (i.e. transaction status P). This reduces the probability of distributed transactions still being in this state at the end of the application and of the application being terminated abnormally with ENDPET.

The following generally applies:
An application involving distributed processing is not terminated normally if, at the time of the abort operation, there are still services with transaction status P (’preliminary end of transaction’) or if confirmations have not yet been received for asynchronous messages to a partner server. UTM then outputs UTM message K060 stating ENDPET as the cause of the abort. No dumps are generated.

Execution / period of validity / transaction management / cluster

The KC_SHUTDOWN call is not subject to transaction management. It cannot be rolled back by an RSET call.

Aborting an application run (KC_KILL) takes immediate effect, there is no return to the program unit.

If the application is to be terminated (KC_NORMAL, KC_WARN and KC_GRACEFUL), the call originates a job, i.e. actions leading to shutdown are initiated.

The shutdown sequence, i.e. how and when UTM terminates the application run is determined by the value specified for *subopcode1* in the parameter area. The shutdown sequence is described in section 1 on page 413.

The following applies in UTM cluster applications:
The effect of the call may be either global to the cluster or local to the node, i.e. the current application run may be terminated at all nodes or only at the node at which the call was issued.

KDCSHUT (page 775)

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area[1]	**identification area**	**selection area**	**data area**
Abort application run immediately (only as dialog)	*subopcode1*: KC_KILL	——	——	—— or *kc_shutdown_str*
Terminate application run normally	*subopcode1*: KC_NORMAL	——	——	—— or *kc_shutdown_str*
Terminate application run normally on expiry of a timer openUTM on a BS2000 system outputs a standard UTM message to all active users)	*subopcode1*: KC_WARN	——	——	*kc_shutdown_str*
Terminate application run normally after expiration of a message and send a UTM message to all active users	*subopcode1*: KC_WARN, *subopcode2*: KC_USER_MSG	——	——	*kc_shutdown_str*
Terminate the application run normally after all UTM-D connections have been cleared, and at the latest after the timer has expired.	*subopcode1*: KC_GRACEFUL	——	——	*kc_shutdown_str*

[1] The operation code KC_SHUTDOWN must always be specified in the parameter area.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_SHUTDOWN
1.	subopcode1	KC_GRACEFUL / KC_KILL / KC_NORMAL / KC_WARN
2.	subopcode2	KC_USER_MSG / —
	id_lth	0
	select_lth	0
3.	data_lth	Length of data in data area / 0

Identification area

—

Selection area

—

Data area

4. Data structure kc_shutdown_str / —

KDCADMI call

```
KDCADMI (&parameter_area, NULL, NULL, &data_area) or
KDCADMI (&parameter_area, NULL, NULL, NULL)
```

Data returned by UTM

Parameter area

	Field name	Content
5.	retcode	Return codes

1. Specify in *subopcode1* how UTM is to terminate the application. You can choose from the following options:

   KC_GRACEFUL
   : UTM prepares for the shutdown. The application is terminated as soon as all UTM-D dialogs have terminated and all UTM D connections have been disconnected or, at the latest, when the specified timer has expired. You must pass the value of the timer in the data area.

     The application is always terminated after the specified timer has expired. If there are no UTM-D connections, the application is immediately terminated normally.

     The following applies after the KC_GRACEFUL call has been processed:

     – It is only possible for users with administration authorization to sign on. Signon attempts from other users will be rejected.
     – It is only possible to call transaction codes for administration programs and the UTM user commands other than KDCOUT. No other services will be started by UTM.
     – All active connections to LPAP and OSI-LPAP partners are set to QUIET.

   KC_KILL
   : The application run is aborted, i.e. it is terminated immediately. Open services are no longer terminated. A UTM dump is created for all processes stating REASON=’ASIS99’.

   KC_NORMAL
   : The application run is terminated normally.
     Shutdown is initiated immediately. The following applies after the KC_SHUTDOWN call:

     – Users/clients can no longer sign on to the application.
     – No further jobs are accepted from partner servers. Users/clients which are already signed on cannot start any new services.
     – New dialog inputs are no longer processed. If the new dialog input is part of a multi-step transaction, the multi-step transaction is rolled back to the last synchronization point.
     – All logical connections to clients, printers and partner applications are shut down.

     Open services can be further processed after the next application start.

KC_WARN

UTM prepares for shutdown. The application is terminated once the specified timer has expired. You must pass the timer value in the data area. The following applies once the KC_SHUTDOWN call has been processed:

– Only users having administration privileges can sign on. Sign-on attempts by other users are rejected

– Only administration program transaction codes and UTM user commands other than KDCOUT can still be called. UTM will no longer start any other services.

– All active connections to LPAP and OSI-LPAP partners are set to QUIET.

2. *subopcode2* is only relevant if it specifies *subopcode1*=KC_WARN. In any other case, nothing may be specified in *subopcode2*.

   Specify *subopcode2*= KC_USER_MSG if UTM is to send a message to all currently active users in preparation for shutdown. You must pass the message which UTM is to send in the data area.

X/W
X/W

The message is accepted in UTM applications on Unix, Linux and Windows systems, but no warning messages are output.

B
B
B

If you do not specify *subopcode2* with KC_WARN on BS2000, all active users are informed by a standard UTM message of the forthcoming shutdown and the time remaining until shutdown.

3. Specify in the *data_lth* field, the length of the data area which you are passing to UTM.

   – for *subopcode1*=KC_KILL, KC_NORMAL:
   No data is passed to/from UTM in the data area (*data_lth*=´0´), or the length of the data structure *kc_shutdown_str* which you pass in the data area.

   – for *subopcode1*= KC_GRACEFUL, KC_WARN:
   Specify in the *data_lth* field the length of the data structure *kc_shutdown_str* which you are passing to UTM in the data area.

4. For *subopcode1*=KC_WARN and *subopcode1*=KC_GRACEFUL, you must pass the data structure *kc_shutdown_str* to UTM in the data area. *kc_shutdown_str* must contain the size of the timer and, if *subopcode2*= KC_USER_MSG, the message to be sent as a warning to all terminal users.

   In the case of standalone UTM applications, values only need to be entered for KC_WARN and KC_GRACEFUL in the data area. The field *scope* in *kc_shutdown_str* is not evaluated.

The following applies in UTM cluster applications:
For each *subopcode1*: In the data structure *kc_shutdown_str*, you can use the *scope* field to control whether only the local node application is to be terminated or whether you want to terminate the entire UTM cluster application, i.e. all the node applications. If you want to initiate a global shutdown of the UTM cluster application, you must enter *scope*=´G´ in the data structure *kc_shutdown_str* . If you do not specify any data structure in the cluster then a local shutdown is performed.

The data structure *kc_shutdown_str* has the following structure:

struct kc_shutdown_str

```
char time_min[3];
char user_message[74];
char scope;
```

time_min Specify in *time_min* the time in minutes after which UTM is to terminate the application run normally.

You should specify a time that is greater than the maximum period that a distributed transaction remains in the state PTC (i.e. transaction status P). In job receiver services, this is the time generated with MAX PTCTIME and in LU6.1 job submitter services, it is the generated time *time2* of the WAITTIME operand in the employed LTAC.

Minimum value: ´1´
Maximum value: ´255´

The entry *time_min*=´0´ is rejected by UTM. If the application is to be terminated normally without any delay, you must specify *subopcode1*=KC_NORMAL.

B Features specific to UTM applications on BS2000 systems:

B B *time_min* is always output to active terminals together with the shutdown warning.

B B B B B B B B B In large UTM applications on BS2000 systems (configurations with many clients), UTM requires a certain amount of time to output the shutdown notice. The selected value of *time_min* should thus not be too small.
In addition, you should define a sufficiently large value for *cpu_time_msec* (see *kc_tac_str* on page 547) for the transaction code by means of which the program unit is started with this KC_SHUTDOWN call. *cpu_time_msec* specifies the maximum CPU time which the program unit run may take up. If the time selected is too short, the shutdown may be aborted.

user_message

Only relevant for *subopcode2*=KC_USER_MESSAGE. If no *subopcode2* was specified, this area is ignored.

Using *user_message* you can pass your own message which UTM is to send to all terminal users as a warning before shutdown. Maximum message length is 74 characters.

B B B B

openUTM on BS2000 systems:
If you do not pass your own warning message in *user_message*, UTM outputs UTM message K023 with the following inserts to all terminal users currently connected to the application:

B B

´hour´:´minutes´:´seconds´
APPLICATION ´name´ WILL BE TERMINATED IN ´minutes´ MINUTES

X/W X/W

openUTM on Unix, Linux and Windows systems:
No warning messages are output on Unix, Linux and Windows systems.

scope Determines whether the local node application is terminated or the entire UTM cluster application, i.e. all the node applications. *scope* is only evaluated for UTM cluster applications.

´L´ Only the local node application is terminated.

´G´ All the node applications in the cluster and therefore also the entire UTM cluster application are terminated.

5. UTM writes the return codes for the call to the *retcode* field. In addition to the return codes listed in section “Return codes” on page 174, the following codes may also occur:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_NOT_ALLOWED *subopcode1* = KC_KILL has been used in an asynchronous service.
KC_SC_NO_GLOB_CHANG_POSSIBLE The generation of the node applications is not currently consistent. You should first shut down the node applications with an old generation..

Main code = KC_MC_DATA_INVALID A field in the data structure in the data area contains an invalid value. **Subcode:**
KC_SC_INVALID_MOD Only for *subopcode1*=KC_GRACEFUL and *subopcode1*=KC_WARN: The application run was not terminated because the time specified in *time_min* is invalid.
Maincode = KC_MC_REJECTED_CURR The call cannot be processed at present. **Subcode:**
KC_SC_INVDEF_RUNNING Only in UTM cluster applications: An inverse KDCDEF is currently running, i.e. the job cannot be processed at present.

Maincode = KC_MC_RECBUF_FULL Only in UTM cluster applications: **Subcode:**
KC_SC_NO_INFO The buffer containing the restart information is full (see openUTM manual “Generating Applications”, KDCDEF control statement MAX, parameter RECBUF).

11.2.14 KC_SPOOLOUT - Establish connections to printers

Using KC_SPOOLOUT you can establish connections to printers. You can:

- establish connections to all printers for which there are print jobs in the associated message queue and to which no connection yet exists.
- establish a connection to the printers which are assigned to a certain LTERM partner. The name of the LTERM partner must be passed in the identification area.

Execution / transaction management / cluster

The KC_SPOOLOUT call is not subject to transaction management. It cannot be rolled back by an RSET call.

Connection setup is triggered by the call, i.e. a job is merely initiated; this fact, however, gives no information as to whether and when a connection will actually be established. You can subsequently ascertain the existence of the connection with an information query (e.g. KC_GET_OBJECT with *obj_type*=KC_LTERM).

The following applies in UTM cluster applications:
The call applies locally to the node, i.e. the connections to the printers are only established in the node application at which the call is issued.

Duration of a connection

Connections to printers for which no print level (PLEV) has been defined remain in existence until they are shut down explicitly (see KC_MODIFY_OBJECT) or the application run is terminated. Connections to printers for which a print level has been defined (PLEV > 0) are shut down after printing.

Using KDCAPPL SPOOLOUT=ON (page 676) you can establish connections to all printers for which print jobs exist.

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identification area**	**selection area**	**data area**
Establish a connection to a printer or to the printers of a printer pool	*obj_number*: 1	Name of the LTERM partner assigned to the printer or printer pool	——	——
Establish connections to all currently unconnected printers for which there are print jobs	*obj_number*: 0	——	——	——

[1] The operation code KC_SPOOLOUT must always be stated in the parameter area.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_SPOOLOUT
1.	obj_number	1 / 0
2.	id_lth	Length of object name in identification area / 0
	select_lth	0
	data_lth	0

Identification area

3.	Object name / —

Selection area

—

Data area

—

KDCADMI call

KDCADMI (&parameter_area, &identification_area, NULL, NULL) or KDCADMI (&parameter_area, NULL, NULL, NULL)

Data returned by UTM

Parameter area

	Field name	Content
4.	retcode	Return codes

1. The values specified in *obj_number* have the following meanings:
   - *obj_number* = 0:
     UTM is to establish a connection to all printers to which connection currently exists and for which there are print jobs.
   - *obj_number* = 1:
     UTM is to establish a connection to the printer or printer pool assigned to a certain LTERM partner. You must pass the name of the LTERM partner in the identification area.
2. You must specify in *id_lth* the length of the object name which you are passing to UTM in the identification area.
   - for *obj_number* = 0 you should specify *id_lth* = 0.
   - for *obj_number* = 1 you should specify in *id_lth* the length of the name which is passed in the identification area.
3. The information you must specify in the identification area is determined by *obj_number*:
   - *obj_number* = 0:
     You may not specify any object name in the identification area. In the KDCADMI call you must pass the null pointer.
   - *obj_number* = 1:
     In the identification area, pass the name of the LTERM partner assigned to the printer or printer pool. To do this, assign the union *kc_id_area* through the identification area and pass the name of the LTERM partner in the *kc_name8* field.

4. UTM writes the return codes for the call to the *retcode* field. In addition to the return codes listed in section “Return codes” on page 174, the following codes may also occur:

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NOT_EXISTENT The LTERM partner specified in the identification area does not exist.
KC_SC_NOT_ALLOWED The operation is not allowed for the stated LTERM partner. Possible reasons are: – the LTERM partner is a dialog partner, i.e. it is not defined for printers (*usage_type* ≠ ´O´) – no printer/printer pool is currently assigned to the LTERM partner – the LTERM partner or the associated printer is currently disabled – the printer belonging to the LTERM partner has been deleted from the configuration – there are no messages for the specified printer, i.e. the LTERM partner’s message queue is empty.
KC_SC_DELETED The specified LTERM partner has been deleted from the configuration.

11.2.15 KC_SYSLOG - Administer the system log file

Using KC_SYSLOG you can administer the system log file SYSLOG during operation. The extent of the functions available to you to administer SYSLOG is determined by whether SYSLOG was created as a simple file or as a file generation group (BS2000 systems) or file generation directory (Unix, Linux and Windows systems). The abbreviation FGG (**F**ile **G**eneration **G**roup) is used hereafter to refer to both file generation directories and file generation groups.

See also the openUTM manual “Generating Applications” and the relevant openUTM manual “Using openUTM Applications” in relation to SYSLOG.

The following functions are available to you, irrespective of whether SYSLOG is maintained as a simple file or as an FGG:

- Write the content of the UTM-internal message buffer to SYSLOG.

  This function is useful if the SYSLOG file, which was created as a simple file, is to be evaluated during operation. All UTM messages with the destination SYSLOG that have been generated by UTM up to this time are then taken into account in the evaluation.

  If SYSLOG was created as an FGG, the following applies:
  When SYSLOG switches over to the next file generation, UTM automatically writes the UTM message buffer to the “old” SYSLOG file generation before switching.

- Have information about the SYSLOG file displayed.

You can also use the following functions if SYSLOG was created as an FGG:

- Activate and deactivate automatic SYSLOG size control.

  Automatic size control means that UTM automatically switches SYSLOG over to the next file generation of the SYSLOG FGG as soon as the size of the current SYSLOG file generation exceeds a certain control value.

- Modify the control value for size monitoring.

- Switch SYSLOG over to the next file generation of the SYSLOG FGG.

SYSLOG size control can even be activated if SYSLOG was not generated with KDCDEF.

Procedure when switching SYSLOG to another file generation

Before switching over to a new file generation, UTM writes the UTM messages still stored in the internal UTM message buffer to the old file generation. All UTM messages generated before switching over are thus written to the “old” SYSLOG. UTM ensures that UTM messages generated after the switch-over time (successful execution of the KC_SYSLOG call) are no longer written to the “old” SYSLOG file generation.

B The following should be noted in UTM applications on BS2000 systems:

B It is possible that the old file generation may not be available immediately after switch-over (i.e. successful processing of the KC_SYSLOG call). The old file generation may still be kept open for a relatively long period by UTM processes, e.g. because the processing of a program unit which was started before the switchover has not yet been concluded and no UTM message with the UTM message destination SYSLOG has yet been written from the associated process.

B Using subopcode1=KC_INFO, you can enquire which SYSLOG file generations have already been closed by all UTM processes. These are all file generations that have a generation number of less than *lowest_open_gen* (see *kc_syslog_str* on page 430).

Period of validity / transaction management / cluster

The call is not subject to transaction management. It takes immediate effect, and the operations initiated by the call will already have been performed when control is returned to the program unit. The call cannot be rolled back.

Modifications to the SYSLOG file size threshold remain in effect until the end of the application run.

If the base of the SYSLOG FGG is within the valid range for the SYSLOG FGG (between the first and last file generation), UTM initially logs in the base file generation in the next application run. If the base is outside the valid range, UTM creates a new file generation for logging as of the next start. The base is specified in the data structure *kc_syslog_str* in the *base_gen* field.

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. the system log file SYSLOG is administered for each node application. Size monitoring persists beyond the current UTM cluster application run. Switching or writing of the buffer apply only to the current UTM cluster application run, i.e. to all the node applications that are currently running.

KDCSLOG (page 779)

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	**identification area**	**selection area**	**data area**
Provide information about SYSLOG	*subopcode1*: KC_INFO *data_lth*: Length of the data area for the return from UTM	——	——	—— (when the call is made you must pass the pointer to a data area for the returns from UTM (*kc_syslog_str)*.)
Set or modify the control value for automatic size control	*subopcode1*: KC_CHANGE_SIZE *data_lth*: length of the data in the data area	——	——	Data structure *kc_syslog_str* with the new control value
Switch SYSLOG over to the next file generation of the FGG	*subopcode1*: KC_SWITCH *data_lth*: 0	——	——	——
Modify the control value for automatic size control and switch SYSLOG over to the next file generation of the FGG	*subopcode1*: KC_SWITCH_AND_CHANGE *data_lth*: Length of the data in the data area	——	——	Data structure *kc_syslog_str* with the new control value
Write UTM message buffer to SYSLOG	*subopcode1*: KC_WRITE_BUFFER *data_lth*: 0	——	——	——

[1] The operation code KC_SYSLOG must always be specified in the parameter area.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_SYSLOG
1.	subopcode1	KC_INFO / KC_CHANGE_SIZE / KC_SWITCH / KC_SWITCH_AND_CHANGE / KC_WRITE_BUFFER
	id_lth	0
	select_lth	0
2.	data_lth	Length of the data structure / length of the data area / 0

Identification area

—

Selection area

—

Data area

3.	Data structure kc_syslog_str / —

KDCADMI call

```
KDCADMI (&parameter_area, NULL, NULL, &data_area)
KDCADMI (&parameter_area, NULL, NULL, NULL)
```

Data returned by UTM

Parameter area

	Field name	Content
4.	retcode	Return codes
5.	data_lth_ret	Length of the data supplied in the data area

Data area

6.	Data structure kc_syslog_str

1. You must specify the operation UTM is to perform in the *subopcode1* field. You can specify the following subopcodes:

   KC_WRITE_BUFFER
   : All UTM messages output with a SYSLOG message destination and which are still stored in the UTM-internal message buffer are immediately written to the current SYSLOG file. If the buffer is empty, the call has no effect.

   KC_INFO Specify if UTM is to return information about the SYSLOG file or SYSLOG FGG. In this case, you must specify in the *data_lth* field the length of the data area which you are making available to UTM to pass the information. For the KDCADMI call you must pass the pointer to this data area.

   You may specify the following values for *subopcode1* only if SYSLOG was created as an FGG.

   KC_CHANGE_SIZE
   : Specify whether you want:

   – to modify the control value for automatic size control. You must pass the threshold in the data area.

   – to activate automatic size control. To do this, pass a control value of > ‘0’ in the data area.

   – to deactivate automatic size control. To do this, pass the control value ‘0’ in the data area.

   KC_SWITCH
   : Specify whether UTM is to switch the SYSLOG file over to the next file generation. If this file generation does not yet exist, UTM creates it.

   KC_SWITCH_AND_CHANGE
   : Corresponds to a combination of the functions of KC_CHANGE_SIZE and KC_SWITCH. Using KC_SWITCH_AND_CHANGE you can switch SYSLOG over to the next file generation and simultaneously modify the

control value for automatic size control. UTM ensures in this case that either both operations are performed successfully or neither is performed; i.e. only if SYSLOG switching was successful does UTM set the new control value.

If UTM cannot switch over to the following file generation, the control value is not modified. Size control is suspended and UTM ignores the new control value. Size control can be reactivated only by a subsequent successful switch-over attempt (repeated KC_SYSLOG call). If a new control value was not specified, UTM carries over the “old” control value.

2. Specify the following in the *data_lth* field.
   - for *subopcode1*=KC_INFO:
     the length of the data area to which UTM is to return the information. When calling KDCADMI, you must pass the pointer to the data area to UTM.
   - for *subopcode1*= KC_CHANGE_SIZE or KC_SWITCH_AND_CHANGE:
     the length of the data in the data area which you are passing to UTM. Pass the data structure *kc_syslog_str* with the new size control value in the data area.
   - for *subopcode1*= KC_SWITCH or KC_WRITE_BUFFER:
     data_lth =0.
     When calling KDCADMI you must specify the null pointer for *&data_area*.
3. The information which you must specify in the data area is determined by *subopcode1:*
   - *subopcode1*=KC_WRITE_BUFFER or KC_SWITCH:
     You must not pass any data to UTM in the data area.
   - *subopcode1*=KC_INFO:
     You may not pass any data to UTM in the data area. You must, however, make a data area available to UTM to which it can return the requested information.
   - *subopcode1*=KC_CHANGE_SIZE or KC_SWITCH_AND_CHANGE:
     You must pass the data structure *kc_syslog_str* with the new control value to UTM in the data area.
     Specify the control value in the *size_control_utmpages* field. The value is specified as the number of UTM pages. Permitted values are between 0 and 2^{31}-1 (specified as char). However, UTM automatically replaces values of between ‘1’ and ‘99’ with ‘100’.
     By using *size_control_utmpages* = ´0´ you deactivate automatic size control.
     You must complete the remaining fields of *kc_syslog_str* with binary zeroes.
     kc_syslog_str is described on page 430.

4. UTM writes the return code for the call to the *retcode* field. In addition to the return codes listed in section “Return codes” on page 174, the following codes may also occur:

Main code = KC_MC_OK The call was processed without errors. **Subcodes:**
KC_SC_MIN_SIZE For *subopcode1* = KC_CHANGE_SIZE or KC_SWITCH_AND_CHANGE: While the size control value was indeed modified, the value specified in *size_control_utmpages* was too low (< 100). The minimum control value of 100 UTM pages was thus set.
KC_SC_BUFFER_EMPTY For *subopcode1* = KC_WRITE_BUFFER: The UTM message buffer is empty and is thus not written to SYSLOG.
KC_SC_SWITCHED The UTM message buffer could not be written to SYSLOG until SYSLOG had been switched to a new file generation.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcodes:**
KC_SC_NO_FGG The requested operation cannot be performed as SYSLOG was created as an FGG.
KC_SC_NO_INFO The operation cannot be performed.
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration changes are possible since the generation of the node applications is not consistent at present.

Main code = KC_MC_DATA_INVALID A field in the data structure in the data area contains an invalid value. **Subcodes:**
KC_SC_INVALID_MOD For *subopcode1* = KC_CHANGE_SIZE or KC_SWITCH_AND_CHANGE: The size control value specified in *size_control_utmpages* is invalid (number too high or no number or not printable). The control value has thus not been modified.
KC_SC_DATA_MISSING For *subopcode1* = KC_CHANGE_SIZE or KC_SWITCH_AND_CHANGE: No size control value was specified in *size_control_utmpages*. The control value has thus not been modified and (for KC_SWITCH_AND_CHANGE) SYSLOG has not been switched.

Main code = KC_MC_DATA_INVALID A field in the data structure in the data area contains an invalid value. **Subcodes:**
KC_SC_DATA_NOT_NULL For *subopcode1* = KC_CHANGE_SIZE or KC_SWITCH_AND_CHANGE: A field that cannot be set in the data structure *kc_syslog_str*, was not supplied with binary zeroes.

Maincode = KC_MC_RECBUF_FULL **Subcode:**
KC_SC_NO_INFO The buffer containing the restart information is full (see openUTM manual “Generating Applications”, KDCDEF control statement MAX, parameter RECBUF).

Maincode = KC_MC_REJECTED_CURR The call cannot be processed at present. **Subcode:**
KC_SC_INVDEF_RUNNING Only in UTM cluster applications: An inverse KDCDEF is currently running, i.e. the job cannot be processed at present.

5. *data_lth_ret* contains the lengths of the data which UTM returns to the data area.
   - for *subopcode1*=KC_INFO returns the information about SYSLOG in the data area (*kc_syslog_str*). *data_lth_ret* ≠ 0 applies.
   - If the length in *data_lth_ret* is less than the data area provided (*data_lth*), the content of the data area is only defined in the length *data_lth_ret*.
   - for *subopcode1*≠ KC_INFO *data_lth_ret* = 0 applies

6. Where *subopcode1*=KC_INFO, UTM returns the data structure *kc_syslog_str* with information about SYSLOG to the application in the data area. The data structure has the following fields:

```
struct kc_syslog_str
char file_name[54];
char curr_size_utmpages[10];
char curr_size_kbyte[10];
char curr_size_percent[3];
char fgg;
char last_switch_ok;
char size_control_engaged;
char size_control_suspended;
char size_control_utmpages[10];
char size_control_kbyte[10];
char start_gen[4];
char curr_gen[4];
char lowest_open_gen[4];
char base_gen[4];
char first_valid_gen[4];
char last_valid_gen[4];
```

The data structure fields have the following meanings:

file_name Name of the current SYSLOG file or file generation in which logging is currently being performed.

curr_size_utmpages
Contains the current size of the SYSLOG file or file generation in which logging is currently being performed. The size is specified as the number of UTM pages occupied by the file or file generation.

curr_size_kbyte
Contains the current size of the SYSLOG file or file generation in which logging is currently being performed. The size is specified in kbytes.

curr_size_percent
If automatic size control is activated, *curr_size_percent* contains the percentage utilization of the SYSLOG file relative to the specified size control value. If size

control has been suspended by UTM or deactivated by means of administration functions, utilization of the SYSLOG file can exceed 100%. In this case, UTM returns blanks in *curr_size_percent*.

If size control has not been defined (either by generation or by means of administration functions), UTM fills *curr_size_percent* with blanks.

fgg Indicates whether SYSLOG was created as an FGG or as a simple file.

´Y´ SYSLOG was created as an FGG.

´N´ SYSLOG was created as a simple file.

All the following items of information are only relevant if SYSLOG was created as an FGG. If SYSLOG was created as a simple file, the following fields will not contain any relevant information.

last_switch_ok
States whether UTM's last attempt to switch over to the next file generation executed without errors. This relates only to switching attempts within the current application run. The following values are possible:

´Y´ The last switch attempt executed without errors.

´N´ An error occurred during UTM's last switch attempt.
UTM could not switch to the next file generation.

´ ´ (Blank)
No switch attempt has yet been made in the current application run or SYSLOG was not created as an FGG.

size_control_engaged
States whether automatic size control is activated. The following values are possible:

´Y´ Size control is activated

´N´ Size control is deactivated

size_control_suspended
States whether automatic size control has been suspended by UTM.

´Y´ The last attempt to switch over to another file generation failed. Size control has, accordingly, been suspended. UTM no longer attempts to switch over to the next file generation even if the defined size control value is exceeded.

Remedy:
You can explicitly attempt to switch the SYSLOG. If switching proceeds without error, size control is reactivated by UTM.

´N´ Size control is not suspended.

size_control_utmpages
: Contains the control value set for automatic size control. The control value is output as the number of UTM pages.

  size_control_utmpages = ´0´ means that size control is deactivated.

  For *subopcode1* = KC_CHANGE_SIZE and KC_SWITCH_AND_CHANGE, pass the new size control value in *size_control_utmpages*.

  Minimum value: ´0´
  Maximum value: 2^{31} -1 (specified as char)

  If you specify *size_control_utmpages* = ´0´, automatic size control is deactivated. UTM automatically replaces values between ‘1’ and ‘99’ with ‘100’.

size_control_kbyte
: Contains the control value set for automatic size control. The control value is output in kilobytes. For very large thresholds, the kilobyte value is not displayed (e.g. for 2^{31} kb).

  size_control_kbyte = 0 means that size control is deactivated.

start_gen Contains the number of the first SYSLOG file generation written by UTM in the current application run.

curr_gen Number of the file generation in which UTM is currently logging data.

lowest_open_gen
: Contains the number of the oldest SYSLOG file generation which is still kept open by an application process.

base_gen Generation number of the defined base for the SYSLOG FGG.

first_valid_gen
: Number of the first valid file generation of the SYSLOG FGG.

B B On BS2000 systems, this corresponds to the specification FIRST-GEN from the SHOW-FILE-ATTRIBUTES command.

last_valid_gen
: Generation number of the last valid file generation of the SYSLOG FGG.

B B On BS2000 systems, this corresponds to the specification LAST-GEN from the SHOW-FILE-ATTRIBUTES command.

11.2.16 KC_UPDATE_IPADDR - Update IP addresses

With KC_UPDATE_IPADDR, while the UTM application is running, you can update the IP addresses stored in the application's object tables using the IP addresses in the hostname database. The host name database that applies to your system can be the hosts file (on Unix, Linux and Windows systems), the DNS (domain name service) or on BS2000 systems the processor table and the socket host table.

B
B
The prerequisite for a comparison on BS2000 systems is that the SOCKET protocol type is generated for the partner or partners.

UTM stores the IP addresses of the following communication partners in the UTM application:

- communication partners that use the socket interface (transport protocol SOCKET) to communicate with the UTM application. These communication partners are generated as clients of the type SOCKET (partner type KC_PTERM).

X/W
X/W
X/W
- communication partners that use the transport protocol RFC1006 to communicate with the application. These can be clients (KC_PTERM with type=APPLI or UPIC-R), LU6.1 partner applications (KC_CON) or OSI TP partner applications (KC_OSI_CON).

For further information on communication using the socket interface and the communication via RFC1006, see the openUTM manual "Generating Applications".

Each time the application is started, UTM reads the IP addresses of the communication partners from the name service and stores them in the object tables.

If the IP addresses of the relevant communication partners change while the application is running, you can request a dynamic update with KC_UPDATE_IPADDR.

With KC_UPDATE_IPADDR you can carry out the following operations:

- update the IP address of a specific communication partner using the name service.
- update the IP address of all communication partners using the name service.

In order to check, you can query the IP addresses stored for the communication partners in the UTM application using KC_GET_OBJECT. UTM returns the IP address in the field *ip_addr* of the data structure of the object type (*kc_con_str*, *kc_osi_con_str* or *kc_pterm_str*).

Execution / period of validity / transaction management / cluster

The job is not subject to transaction management. It takes immediate effect and the IP addresses will already be updated on return to the program unit. The job cannot be undone.

The IP addresses updated with KC_UPDATE_IPADDR remain stored in the UTM application until the application is terminated or until KC_UPDATE_IPADDR is next applied within the current application run.

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. the IP address update is performed at all currently running node applications.

Data to be supplied

Function of the call	**Data to be entered in the**			
	parameter area [1]	**identification area**	**selection area**	**data area**
Update IP addresses of a communication partner	*subopcode1*: KC_PARTNER *obj_type*: partner types *obj_number*: 1	Union *kc_id_area* with the name or triad of names of the partner	——	1. Pointer to the data area in which UTM returns the data structure of the object type with the new IP address.
Update the IP addresses of all communication partners concerned with the database for the host names	*subopcode1*: KC_ALL *obj_type*: KC_NO_TYPE *obj_number*: 0	——	——	——

[1] In all cases the operation code KC_UPDATE_IPADDR must be supplied in the parameter area.

Parameter settings

Parameter area

	Field name	Content
	version	KC_ADMI_VERSION_1
	retcode	KC_RC_NIL
	version_data	KC_VERSION_DATA
	opcode	KC_UPDATE_IPADDR
1.	subopcode1	KC_PARTNER / KC_ALL
2.	obj_type	KC_CON / KC_OSI_CON / KC_PTERM / KC_NO_TYPE
3.	obj_number	1 / 0
4.	id_lth	Length of the partner name / 0
	select_lth	0
5.	data_lth	Length of the data area / 0

Identification area

6.	Partner name / —

Selection area

—

Data area

7.	Data structure of the object type / —

KDCADMI call

KDCADMI (&parameter_area, &identification_area, NULL, &data_area) or
KDCADMI (&parameter_area, NULL, NULL, NULL)

Data returned by UTM

Parameter area

	Field name	Content
8.	retcode	Return code
9.	data_lth_ret	Length of the date returned in the data area / 0

Data area

10.	Data structure of the object type/ —

1. In the field *subopcode1* you must specify:

   KC_PARTNER
   if UTM is to update the IP address of a specific communication partner. Pass the name of the partner in the identification area.

   KC_ALL if UTM is to update the IP addresses of all communication partners that communicate with the UTM application using the appropriate protocol with the data in the host name database.
   The appropriate protocol types are:

   – SOCKET
   – RFC1006

2. In the field *obj_type* you must specify the object type of the communication partner.

   With *subopcode1*=KC_ALL you must specify *obj_type*=KC_NO_TYPE.

   With *subopcode1*=KC_PARTNER you can make any of the following entries:

   KC_PTERM
   for partner applications configured as clients of the following type
   – SOCKET (BS2000 systems)
   – APPLI, UPIC-R or SOCKET (Unix, Linux and Windows systems)

   KC_CON for an LU6.1 partner application

   KC_OSI_CON
   for an OSI TP partner application

3. In *obj_number* you must specify the number of objects for which the IP address is to be updated.

   – for *subopcode1*=KC_PARTNER you must enter *obj_number*=1

   – for *subopcode1*=KC_ALLyou must enter *obj_number*=0. UTM will then update the IP address of all communication partners with the relevant configuration.

4. Which entries you must make in the field *id_lth* depends on the entry in the field *subopcode1*:
   - for *subopcode1*=KC_PARTNER:
     you must enter the length of the data structure in *id_lth* which you pass to UTM in the identification area.
   - for *subopcode1*=KC_ALL:
     you must set *id_lth*=0.
5. In the field *data_lth* you enter the length of the data area. You must make the following entries:
   - for *subopcode1*=KC_PARTNER:
     length of the data structure of the object type in *obj_type*.
   - for *subopcode1*=KC_ALL:
     data_lth=0.
6. Which data you must supply to the identification area depends on *subopcode1*.
   - for *subopcode1*=KC_PARTNER:
     In the identification area, you must supply the union *kc_id_area* and the name of the communication partner. The entry must identify the partner unambiguously.

     for *obj_type*=KC_PTERM you must supply the name triad comprising client name (PTERM), the processor name and the BCAMAPPL name in the *kc_triple_str* structure of the union.

     X/W X/W X/W for *obj_type*=KC_CON you must supply the name triad comprising the application name, the processor name and the BCAMAPPL name in the structure *kc_triple_str* of the union.

     X/W X/W for *obj_type*=KC_OSI_CON you must enter the name of the connection to the OSI TP partner application in the *kc_name8* field of the union.
   - for *subopcode1*=KC_ALL you must pass the null pointer.
7. What values you enter in the data area depends on *subopcode1*:
   - for *subopcode1*=KC_PARTNER specify the data structure of the object type (*kc_con_str*, *kc_osi_con_str* or *kc_pterm_str*).
   - for *subopcode1*=KC_ALL you must pass the null pointer.

8. In the field *retcode* UTM supplies the return code of the call. Beside the return codes listed in section “Return codes” on page 174, the follow return codes can also occur:

Maincode = KC_MC_REJECTED UTM rejected the call. **Subcodes:**
KC_SC_TPROT_NOT_ALLOWED This transport protocol is not supported, i.e. no communication partners for communication via SOCKET are generated in the application. This return code can also occur when, although a communication partner is generated in the application for communication via SOCKET (e.g. BCAMAPPL), KC_PARTNER is specified with the object type KC_CON or KC_OSI_CON. On BS2000 systems it is only possible to specify the object type KC_PTERM for KC_PARTNER. This code is also returned if at least one communication partner and the associated BCAMAPPL with T-PROT=SOCKET has not been generated in the application.
KC_SC_SOCKET_ERROR It was not possible to update the IP address(es) due to an error in the communication interface (socket call).
KC_SC_INVALID_NAME The communication partner specified in the identification area does not exist or it does not use the required transport protocol for communication with UTM.
KC_SC_NO_IPADDR_FOUND *subopcode1*=KC_PARTNER: No IP address was found for the specified communication partner in the name service. *subopcode1*=KC_ALL: UTM did not find an IP address for any of the communication partners of the specified object type in the name service
KC_SC_AT_LEAST_ONE_OBJ_FAILED The IP addresses have been compared using *subopcode1*=KC_ALL. However, an error occurred with at least one object. This may possibly be caused by errors described in the previous return codes. You will find information on the partner(s) on which an error occurred in message K154, which is by default output to SYSLOG and SYSOUT.
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration is possible since the generation of the node applications is not consistent at present.

Maincode = KC_MC_RECBUF_FULL **Subcode:**
KC_SC_NO_INFO The buffer containing the restart information is full (see openUTM manual “Generating Applications”, KDCDEF control statement MAX, parameter RECBUF).

Maincode = KC_MC_REJECTED_CURR The call cannot be processed at present. **Subcode:**
KC_SC_INVDEF_RUNNING Only in UTM cluster applications: An inverse KDCDEF is currently running, i.e. the job cannot be processed at present.

9. *data_lth_ret* contains the length of the data returned by UTM in the data area.
   - for *subopcode1*=KC_PARTNER: length of the data returned by UTM in the data area
   - for *subopcode1*=KC_ALL: *data_lth_ret*=0
10. For *subopcode1*=KC_PARTNER UTM returns the data structure of the object type (*kc_con_str, kc_osi_con_str* or *kc_pterm_str*) in the data area with the following information:
    - If the new IP address of the communication partner is an IPv4 address, it is located in the *ip_addr* field of the data structure and has a length of 15. The *ip_v* field contains V4.
    - If the new IP address of the communication partner is an IPv6 address, it is located in the *ip_addr_v6* field of the data structure and has a length of 39. The *ip_v* field contains V6.
    - The other fields of the data structure do not contain any information.

11.2.17 KC_USLOG - Administer the user log file

The user log file is managed as the file generation directory USLOG. Using KC_USLOG, you can close the current user log file (file generation of USLOG) and simultaneously open a new user log file, which is the file generation with the next generation number. The closed log file may then be put to any use you require.

Switching with dual USLOG

If the user log file of your application is operated with dual files (see the openUTM manual “Generating Applications”), the KC_USLOG call acts on both files.

Period of validity of the change / cluster

Successful processing of the call means that UTM has successfully switched to the next file generation. UTM writes all LPUT messages generated after the switch to the new log file. After switching, UTM also writes the LPUT messages to the new USLOG file generation(s) until you again switch to the following file generation.

The following applies in UTM cluster applications:
The call applies globally to the cluster, i.e. the current user log file is closed and a new user log file is opened at all currently running node applications.

KDCLOG (page 742)

Data to be supplied

Function of the call	Data to be entered in the			
	parameter area [1]	identifi-cation area	selection area	data area
Switch the user log file over to the next file generation of the FGG	*subopcode1*: KC_SWITCH *data_lth*: 0	——	——	——

[1] The operation code KC_USLOG must be specified in the parameter area.

Parameter settings

Parameter area

Field name	Content
version	KC_ADMI_VERSION_1
retcode	KC_RC_NIL
version_data	KC_VERSION_DATA
opcode	KC_USLOG
subopcode1	KC_SWITCH
id_lth	0
select_lth	0
data_lth	0

Identification area

—

Selection area

—

Data area

—

KDCADMI call

```
KDCADMI (&parameter_area, NULL, NULL, NULL)
```

Data returned by UTM

Parameter area

	Field name	Content
1.	retcode	Return codes

1. UTM writes the return codes for the call to the *retcode* field. In addition to the return codes listed in section “Return codes” on page 174, the following codes may also occur:

Main code = KC_MC_REJECTED_CURR The call cannot be processed at the present time. **Subcode:**
KC_SC_LWRT_IN_PROGRESS USLOG cannot be switched to the next file generation at the present time as UTM is currently writing data to the USLOG.
KC_SC_INVDEF_RUNNING Only in UTM cluster applications: An inverse KDCDEF is currently running, i.e. the job cannot be processed at present.

Main code = KC_MC_REJECTED The call was rejected by UTM. **Subcode:**
KC_SC_FILE_ERROR It is not possible to switch USLOG to the next file generation due to a DMS error.
KC_SC_NO_GLOB_CHANG_POSSIBLE Only in UTM cluster applications: No global administration is possible since the generation of the node applications is not consistent at present.

Maincode = KC_MC_RECBUF_FULL **Subcode:**
KC_SC_NO_INFO The buffer containing the restart information is full (see openUTM manual “Generating Applications”, KDCDEF control statement MAX, parameter RECBUF).

11.3 Data structures used to pass information

The data structures that you must place in the data area after calling KC_GET_OBJECT, KC_MODIFY_OBJECT oder KC_CREATE_OBJECT are described in this section.

- For KC_GET_OBJECT UTM returns the object properties, application parameters and statistical data queried in the format of these data structures. The data structures are defined in the *kcadminc.h* header file.
- The data are passed to UTM in this format when changing object properties and application parameters (KC_MODIFY_OBJECT) and when dynamically adding new objects to the configuration (KC_CREATE_OBJECT).

The following two sections describe the data structures and the meanings of their constituent elements.
The section “Data structures for describing object properties” on page 445 describes the structures used to pass information about objects of the application.
The section “Data structures used to describe the application parameters” on page 591 describes the structures used to pass application parameters.
There are other data structures that do not belong to any object or parameter type in addition to those described in these sections. You will need these for certain calls to pass data to UTM. These data structures are covered in the descriptions of the corresponding operation codes.

Their names are created as follows: *operationscode_str*.

The following data structures belong to this group:

- You need *kc_change_application_str* when passing data for a program replacement with KC_CHANGE_APPLICATION (page 179f).
- You need *kc_create_statements_str* to pass data to UTM when requesting an inverse KDCDEF run with KC_CREATE_STATEMENTS (page 246).
- You need *kc_encrypt_advanced_str* or *kc_encrypt_str* to read the public key of an RSA key pair with KC_ENCRYPT (page 268f).
- You need *kc_shutdown_str* to pass data to UTM when requesting a shutdown with KC_SHUTDOWN (page 409f).
- You need *kc_syslog_str* when administering the SYSLOG file with KC_SYSLOG (page 422f).
- You need *kc_online_import_str* to import application data online with KC_ONLINE_IMPORT.
- You need *kc_lock_mgmt_str* to release locks in UTM cluster applications using KC_LOCK_MGMT.

General information on the structure of the data structures

The fields in the data structures are not all of the data type "char". The square brackets following the name of the field contain the length of the field. If there are no square brackets, then the field is 1 byte long.

The following points must be observed when exchanging data between UTM and an administration program unit:

- Names and keywords are left-justified and are padded to the right with blanks.
  The data passed to UTM must be in uppercase letters, except for object names.

  Example
  The *ptype* (*kc_pterm_str*) field is 8 bytes long. *ptype* =APPLI is stored as follows: ´APPLI␣␣␣´.

- Numeric data is stored right-justified by UTM and is returned with leading blanks. When data is passed from an administration program to UTM, left- and right-justified data is accepted. Right-justified data is accepted with leading blanks or zeros. Left-justified data can also be terminated by the null byte (\0) or padded with blanks.

  Example
  The *conn_users* (*kc_max_par_str*) field is 10 bytes long. *conn_users* =155 is stored by UTM as follows: ´␣␣␣␣␣␣␣155´.

- When passing data to UTM, fields in the data structures in which no values are specified are to be supplied with binary zero.

Description format

The data structures in *kcadminc.h* are presented in tables. The tables have the following structure:

mod	Data structure kc_..._str	Page
1.	2.	3.

1. The first column (shaded gray) specifies which parameters, i.e. field contents, you can modify with KC_MODIFY_OBJECT. If the "mod" column does not contain data, then you cannot modify any parameters.

   The abbreviations used in the first column have the following meanings:

   – The parameter cannot be modified.

   x(*y*) The value of the parameter can be modified.

   The value in brackets (*y*) informs you of how long the modification is effective and in which way. *y* can take on one of the following values: IR/GIR, ID/GID, PR/GPR, PD/GPD, P/GP, A/GA. See page 313 for the meaning of the abbreviations.

2. The second column contains the fields of the data structure as it is defined in *kcadminc.h*.
3. The third column is only used for presenting very large data structures. This column lists the page where you can find the description corresponding to the data structure field.

The meanings of the contents of the fields are described at the end of each table.

11.3.1 Data structures for describing object properties

All data structures provided for passing object properties are described in this section. Each individual object type is provided with a data structure of its own. You will find these data structures in the *kcadminc.h* header file. The name of the data structure is created from the name of the object type and the "*_str*" suffix. The descriptions are listed in alphabetically ascending order according to the names of the data structures.

Data structures can contain *filler* fields at the end. These are not listed here.

kc_abstract_syntax_str - Abstract syntax for communication via OSI TP

The data structure *kc_abstract_syntax_str* is defined for object type KC_ABSTRACT_SYNTAX. With KC_GET_OBJECT, UTM returns the local name, the object identifier and the name of the allocated syntax for an abstract syntax.

In communication using OSI TP, the abstract syntax specifies how the user data is to be encrypted before being transferred to the communication partner.
Both communication partners must use the same abstract syntax.

Data structure kc_abstract_syntax_str

```
char abstract_syntax_name [8];
char object_id[10][8];
char transfer_syntax[8];
```

The fields in the data structure have the following meanings:

abstract_syntax_name
: Contains the local name of the abstract syntax. The name is up to 8 characters in length.

  The local name must be specified in an MGET/MPUT or FGET/FPUT if data with this abstract syntax is to be sent or received.

object_id
: Contains the object identifier of the abstract syntax.

  The object identifier consists of at least 2 and at most 10 components. These components are positive integers between 0 and 67 108 863.

  For each component of the object identifier, UTM returns a field element, i.e. the number of field elements occupied in *object_id* corresponds to the number of components. The other field elements contain binary zeros.

  For further information on the object identifier see the openUTM manual “Generating Applications”.

transfer_syntax
: Contains the local name of the transfer syntax allocated to the abstract syntax.

kc_access_point_str - OSI TP access point

The data structure *kc_access_point_str* is defined for the object type KC_ACCESS_POINT. In the case of KC_GET_OBJECT, UTM returns the name and address of a local OSI TP access point in *kc_access_point_str*.

A local OSI TP access point is statically generated using the KDCDEF control statement ACCESS-POINT.

Data structure kc_access_point_str

```
char ap_name[8];
char application_entity_qualifier[8];
union kc_selector presentation_selector;
union kc_selector session_selector;
char presentation_selector_type;
char presentation_selector_lth[2];
char presentation_selector_code;
char session_selector_type;
char session_selector_lth[2];
char session_selector_code;
char transport_selector[8];
char listener_id[5];
char listener_port[5];
char t_prot[6];
char tsel_format;
```

X/W X/W X/W X/W

The fields in the data structure have the following meanings:

ap_name
: Name of the OSI TP access point. The OSI TP access point is uniquely identified within the local UTM application by this name. *ap_name* is a maximum of 8 characters long.

application_entity_qualifier

The application entity qualifier (AEQ) of the access point. The AEQ is required for addressing purposes when communicating with heterogeneous communication partners. These communication partners address the access point via the application process title (APT) of the local application and the AEQ of the access point. The AEQ is a positive integer between 1 and 67108863 (= 2^{26}-1). You will find more information on the AEQ in the openUTM manual “Generating Applications”.

application_entity_qualifier=´0´ means that no AEQ is defined for the access point.

presentation_selector

Contains the presentation selector for the address of the OSI TP access point.

presentation_selector is a field of type *kc_selector*:

union kc_selector
`char x[32];`
`char c[16];`

UTM generally returns the presentation selector as character string (*c*) in a machine-specific code format (*presentation_selector_code*=´S´). The string is a maximum of 16 characters long. The *presentation_selector* field is padded with blanks starting after the position specified in the *presentation_selector_lth* length field.

In special cases the presentation selector is returned as a hexadecimal string (*x*). Each half byte is represented by a character. For example, the hexadecimal number A2 is returned as the string ´A2´ (2 characters). If the presentation selector is a hexadecimal number, then UTM returns up to 32 bytes.

You determine how to interpret the contents of the *presentation_selector* with the *presentation_selector_type* field.

If the address of the access point does not contain a presentation selector, then the *presentation_selector* field contains only blanks. In this case, *presentation_selector_type* = ´N´ and *presentation_selector_lth* = ´0´.

session_selector

Contains the session selector of the address of the OSI TP access point.

session_selector is a union of type *kc_selector (see presentation_selector*).

UTM generally returns the session selector as character string (*c*) in a machine-specific code format (*session_selector_code*=´S´). The string is a maximum of 16 characters long. The *session_selector* field is padded with blanks starting after the position specified in the *session_selector_lth* length field.

In special cases the session selector is returned as a hexadecimal string (*x*). Each half byte is represented by a character. If the session selector is a hexadecimal number, then UTM returns up to 32 bytes in *session_selector*.

You determine how to interpret the contents of the *session_selector* with the *session_selector_type* field.

If the address of the access point does not contain a presentation selector, then the *session_selector* field contains only blanks. In this case, *session_selector_type* = ´N´ and *session_selector_lth* = ´0´.

presentation_selector_type

Specifies if the address of the access point contains a presentation selector and how to interpret the data returned in *presentation_selector*.

´N´ N stands for *NONE. The address of the access point does not contain a presentation selector, *presentation_selector* contains only blanks and *presentation_selector_lth*=´0´.

´C´ The data of the presentation selector in *presentation_selector* is to be interpreted as a character string. A maximum of the first 16 bytes of *presentation_selector* contain data.

´X´ The presentation selector in *presentation_selector* is a hexadecimal number.

presentation_selector_lth

Contains the length of the presentation selector (*presentation_selector)* in bytes. If *presentation_selector_lth* =´0´, then the address of the OSI TP access point does not contain any presentation components (*presentation_selector* contains blanks). Otherwise, the value of *presentation_selector_lth* lies between ´1´ and ´16´.

If *presentation_selector_type*=´X´, then the string length specified in *presentation_selector* is: 2 ∗ *presentation_selector_lth* bytes.

Example
The presentation selector of the access point is X´A2B019CE´. *presentation_selector* then contains the string ´A2B019CE´, *presentation_selector_type*=´X´ and *presentation_selector_lth* =´ 4´ (four hexadecimal numbers).

presentation_selector_code

Specifies how the presentation selector in *presentation_selector* is encoded.

UTM returns ´S´ if the presentation selector is returned as a character string (*presentation_selector_type* = ´C´).
´S´ means: machine-specific code (default code: EBCDIC on BS2000 systems and ASCII on Unix, Linux and Windows systems).

If *presentation_selector_type* = ´X´ or ´N´, then UTM returns a blank in the *presentation_selector_code* field.

session_selector_type

Specifies if the address of the access point contains a session selector and how to interpret the data returned in *session_selector*.

´N´ N stands for *NONE. The address of the access point does not contain a session selector, *session_selector* contains only blanks and *session_selector_lth*=´0´.

´C´ The data of the session selector in *session_selector* is to be interpreted as a character string. A maximum of the first 16 bytes of *session_selector* contain data.

´X´ The session selector in *session_selector* is a hexadecimal number.

session_selector_lth

Contains the length of the session selector (*session_selector)* in bytes.

If *session_selector_lth* =´0´, then the address of the OSI TP access point does not contain any session components (*session_selector* contains blanks).
Otherwise, the value of *session_selector_lth* lies between ´1´ and ´16´.

If *session_selector_type*=´X´, then the string length specified in *session_selector* is: 2 ∗ *session_selector_lth* bytes.

session_selector_code

Specifies how the session selector in *session_selector* is encoded.

UTM returns ´S´ if the session selector will be returned as a character string (*session_selector_type* = ´C´).
´S´ means: machine-specific code (default code: EBCDIC on BS2000 systems and ASCII on Unix, Linux and Windows systems).

If *session_selector_type* = ´X´ or ´N´, then UTM returns a blank in the *session_selector_code* field.

transport_selector

Contains the transport selector of the address of the OSI TP access point. *transport_selector* always contains a valid value because each access point must be assigned a transport selector in the KDCDEF generation. The transport selector is always to be interpreted as a character string and consists of 1 to 8 printable characters.

B
B The value of *transport_selector* is a local BCAM application name on BS2000 systems.

X/W listener_id

X/W Contains the listener ID of the access point. The listener ID is a positive integer
X/W between 0 and 32767 (default value).

X/W The listener ID determines which connections are to be administered by the same
X/W net process. All connections established via access points and BCAMAPPL names
X/W with the same listener ID will be administered by a single net process.

X/W An exception to this are the BCAMAPPL names for communication via the socket
X/W interface (SOCKET). They form a number set of their own, i.e. access points with
X/W these BCAMAPPL names are not bundled in a single net process, even if the
X/W listener ID is the same.

X/W The following fields are only significant for access points of a UTM application under UNXI
X/W systems and Windows systems. These fields contain the address components of the
X/W access point in the local system. See the openUTM manual “Generating Applications” for
X/W more information.

X/W listener_port

X/W Contains the port number of the access point for establishing TCP IP connections.
X/W The port number specified is the port number defined in the KDCDEF generation.

X/W Possible values of *listener_port*: ´0´, ´1025´ to ´32767´

X/W If *listener_port* = ´0´, then no listener port number was generated for this access
X/W point in the KDCDEF generation.

X/W t_prot Contains the address format assigned to the access point during KDCDEF gener-
X/W ation.

X/W The address formats are specified as follows:

X/W ´R´ RFC1006, ISO transport protocol class 0 using TCP/IP and the conver-
X/W gence protocol RFC1006.

X/W If *t_prot* contains only blanks, then no address format was defined in the KDCDEF
X/W generation.

X/W X/W tsel_format

Contains the format indicator of the T-selectors in the address of the access point.

X/W ´T´ TRANSDATA format

X/W ´E´ EBCDIC character format

X/W ´A´ ASCII character format

X/W X/W If *tsel_format* contains a blank, then no format indicator was defined in the KDCDEF generation.

X/W X/W The meaning of the address format is described in the “PCMX documentation” on page 20.

kc_application_context_str - Application context for communication via OSI TP

The data structure *kc_application_context_str* is defined for object type KC_APPLICATION_CONTEXT. In the case of KC_GET_OBJECT, UTM returns the local name and the properties of an application context in this data structure.

The application context defined the rules governing data communication between the communication partners. It specifies how the user data is coded for transfer (abstract syntax) and in which form the data is transferred (transfer syntax).

The application context must be agreed with the partner. For further information on the application context see the openUTM manual “Generating Applications”.

Data structure kc_application_context_str

```
char application_context_name [8];
char object_id[10][8];
char abstract_syntax[9][8];
```

The fields in the data structure have the following meanings:

application_context_name
: Contains the name generated locally for the application context. It has a maximum length of 8 characters.

object_id
: Contains the object identifier of the application context.

  The object identifier consists of at least 2 and at most 10 components. The individual components are positive integers between 0 and 67108863.

  For each component of the object identifier, UTM returns a field element, i.e. the number of field elements occupied in *object_id* corresponds to the number of components. The other field elements contain binary zeros.

  For further information on the object identifier see the openUTM manual “Generating Applications”

abstract_syntax
: Contains the local names of the abstract syntax allocated to the application context. Up to 9 abstract syntaxes can be allocated to one application context. For each abstract syntax, UTM returns a field element, i.e. the number of occupied field elements in *abstract_syntax* corresponds to the number of abstract syntaxes allocated to the application context. The remaining field elements are filled with binary zeros.

  Each application context is assigned at least one abstract syntax.

kc_bcamappl_str - Names and addresses of the local application

The data structure *kc_bcamappl_str* is defined for the object type KC_BCAMAPPL. In the case of KC_GET_OBJECT, UTM returns the names and properties of the local application in *kc_bcamappl_str*.

UTM informs about the properties of the local application that are assigned tithe application name as defined in MAX APPLI or to the BCAMAPPL names of the application. BCAMAPPL names are also the application names that are used for distributed processing with LU6.1 and for connecting to clients; they are generated with the KDCDEF statement BCAMAPPL. The names assigned to the application are used to establish connections between the communication partners and the application. Each name of the application is assigned its own address for establishing a connection.

Data structure kc_bcamappl_str

```
char bc_name[8];
char t_prot[6];
X/W  char listener_id[5];
char listener_port[5];
X/W  char tsel_format;
char signon_tac[8];
```

The fields in the data structure have the following meanings:

bc_name
: Contains the name of the local application whose properties UTM returns. It is up to 8 characters long.

t_prot
: The meaning of the data returned in *t_prot* depends on the operating system under which the UTM application is running.

B *BS2000 systems:*

B B B *t_prot* contains the transport protocol used for connections to partner applications established using this application name.
Only the first field element of *t_prot* contains data. The rest contain blanks.

B The transport protocol is specified as follows:

B ´N´ NEA transport protocol

B ´I´ ISO transport protocol

B ´R´ TCP/IP together with the convergence protocol RFC1006
B (TCP-RFC1006)

B ´T´ Native TCP/IP communication via the socket interface

X/W *Unix, Linux and Windows systems:*

X/W *t_prot* contains the address format assigned to the BCAMAPPL names during
X/W KDCDEF generation.

X/W Each address format generated occupies an element of the *t_prot* field the rest of
X/W the field elements are filled with blanks.

X/W The address formats are specified as follows:

X/W ´R´ RFC1006, ISO transport protocol class 0 using TCP/IP and the conver-
X/W gence protocol RFC1006.

X/W listener_id
X/W Contains the listener ID of the BCAMAPPL names. The listener ID is a positive
X/W integer between 0 and 32767.

X/W The listener ID determines which connections are to be administered together by
X/W the same net process. All connections established via access points and
X/W BCAMAPPL names with the same listener ID will be administered by a single net
X/W process.

X/W BCAMAPPL names with *t_prot*=´T´ (SOCKET) form a separate set of numbers, i.e.
X/W no BCAMAPPL names for communication via socket interface are bundled with
X/W BCAMAPPL names/access points for other transport protocols in a single net
X/W process, even if the listener ID is the same.

listener_port
Only applies if *t_prot*=´T´ or ´R´.

listener_port contains the port number at which openUTM waits for connection requests from outside. The port number specified at KDCDEF generation is passed. See also the openUTM manual “Generating Applications”.

B In UTM applications on BS2000 systems, *listener_port* is only used if *t_prot*=´T´ is
B generated. In all other cases *listener_port*=´0´.

X/W Possible values of *listener_port*: ´0´, ´1025´ to ´32767´

X/W If *listener_port* = ´0´, then no listener port number was generated.

X/W X/W tsel_format

Contains the format indicator of the T-selector in the address.

X/W ´T´ TRANSDATA format

X/W ´E´ EBCDIC character format

X/W ´A´ ASCII character format

X/W X/W If *tsel_format* contains a blank, then no format indicator was defined in the KDCDEF generation.

X/W X/W The meanings of the address formats are described in the “PCMX documentation” on page 20.

signon_tac

signon_tac either contains the name of the transaction code of the sign-on service assigned to this transport system access point or is empty (no sign-on service).

kc_cluster_node_str - Node applications of a UTM cluster application

The data structure *kc_cluster_node_str* is defined for the parameter type KC_CLUSTER_NODE. In the case of KC_GET_OBJECT, openUTM uses *kc_cluster_node_str* to return the properties of the individual node applications (instances) in a UTM cluster application.

mod[1]	Data structure kc_cluster_node_str
–	`char node_indx[4];`
x(GID)	`char hostname[8];`
x(GID)	`struct kc_file_base filebase;`
x(GID)	`char catid_a[4];`
–	`char bcamappl[8];`
–	`char port_nbr[8];`
–	`struct kc_admi_date_time_model kdcdef_time;`
–	`struct kc_admi_date_time_model startup_time;`
–	`struct kc_admi_date_time_model shut_n_time;`
–	`char start_type;`
–	`char node_state;`
–	`char monitored_node[4];`
–	`char monitoring_node[4];`
–	`struct kc_admi_date_time_model state_change_time;`
x(GID)	`char virtual_host[8];`
–	`node_name[8]`

B

[1] Field content can be modified with KC_MODIFY_OBJECT , see page 311f

The fields in the data structure have the following meanings:

node_indx
: Number (index) of the node application in the UTM cluster application. The number is assigned internally in the cluster and is used for diagnostic purposes. The index uniquely identifies the node application within the UTM cluster application.
The node index is determined on the basis of the sequence of CLUSTER-NODE statements in the KDCDEF input: The node that is described by the first statement to occur has the index ‘1', the second '2' etc.
KC_MODIFY_OBJECT:
In order to modify the properties of a node application, you must pass the number of the node application in the identification area. You may first need to determine the number by means of a KC_GET_OBJECT call. You can only modify nodes that are not active.

hostname
: Contains the primary host name of the node on which this node application is running.

  KC_MODIFY_OBJECT:
  Specify the primary name of the node on which the node application is to run.

  The name can be up to 64 characters in length.

filebase
: Base name of the KDCFILE, the user log file and the system log file SYSLOG for the node application. When the node application is started, the UTM system files are expected under the name specified here.

  The name is passed in the element *filebase* of type *kc_file_base*:

struct kc_file_base
`char length[2];`
`char fb_name[42];`

  fb_name contains the base name and *length* the length of the base name.

  KC_MODIFY_OBJECT:
  You can modify the base name of the node application. When doing so, please note the following:

  – The base names of the individual node applications of a UTM cluster application must differ from each other.

B
B
B
  – *BS2000 systems*:
  Specify the name without the catalog ID. You must specify the catalog ID in the operand *catid_a*.

B
B
The base name may contain a BS2000 user ID and may be up to 42 characters in length.

X/W
X/W
X/W
X/W
– *Unix, Linux and Windows systems:*
Specify the directory which contains the UTM system files of the node application. The name specified here must identify the same directory for all the nodes. It may be up to 27 characters in length.

B
B
B
catid_a
Catalog ID, the UTM system files of the node application are assigned to (in particular the KDCFILE).

bcamappl
Name of the transport system endpoint (BCAMAPPL name) that is used for communication within the cluster. It is defined in the CLUSTER statement during generation.

port_nbr
Number of the listener port used for communication within the cluster. It is defined in the CLUSTER statement during generation.

kdcdef_time
Time at which the KDCFILE of this node application was generated.

The date and time are returned in the element *kdcdef_time* of type *kc_admi_date_time_model*:

struct kc_admi_date_time_model
`struct kc_admi_date_model admi_date;`
`struct kc_admi_time_model admi_time`

where

struct kc_admi_date_model
`char admi_day [2];`
`char admi_month [2];`
`char admi_year_4 [4];`
`char admi_julian_day [3];`
`char admi_daylight_saving_time`

and

struct kc_admi_time_model
`admi_hours [2];`
`admi_minutes [2];`
`admi_seconds [2]`

startup_time
: Time of the last start of this node application.

  The date and time of the start are returned in the element *startup_time* of type *kc_admi_date_time_model* (see *kdcdef_time*).

shut_n_time
: Time at which this node application was last terminated normally.

  The date and time are returned in the element *shut_n_time* of type *kc_admi_date_time_model* (see *kdcdef_time*).

state_change_time;
: Time of the last status change of this node application (see *node_state*).

  The date and time are returned in the element *state_change_time* of type *kc_admi_date_time_model* (see *kdcdef_time*).

start_type

  Type of the last start of this node application:

  ’C’ The last start of the application was a cold start following a normal termination of the application (COLD).

  ’W’ The last start of the application was a warm start following an abnormal termination of the application (WARM).

  ’D’ The node application was started for the first time after the generation run (DEF).

  ’U’ The node application was started after a KDCUPD run (UPDATE).

node_state
: State of the node application:

  ’G’ (Generated)
  : The node application has not yet been started after the generation run.

  ’R’ (Running)
  : The node application is currently running.

  ’T’ (Terminated)
  : The node application is not running. It was terminated normally.

  ’A’ (Abnormally terminated)
  : The node application is not running. It was terminated abnormally.

  ’F’ (Failure)
  : The node application was identified as failed by its monitoring node application.

monitored_node
: Number (index) of the node application which is monitored by this node application, i.e. whose availability is cyclically checked.

monitoring_node
: Number (index) of the node application which monitors the availability of this node application.

virtual_host

B *BS2000 systems:*

B Name of the virtual host, as understood by BCAM, in which the node application
B should run.

B Default value: 8 blanks, i.e. the node application runs under the name of the real
B host

X/W *Unix, Linux and Windows systems:*

X/W By specifying HOSTNAME, it is possible to specify the sender address for network
X/W connections which are established from this node application.

X/W Default value: 8 blanks, i.e. the default sender address of the transport system is
X/W used when connections are established. This function is required in a cluster if the
X/W relocatable IP address is to be used as the sender address instead of the static IP
X/W address when establishing a connection.

node_name
: Reference name of the node application.

B/X Default: NODE*nn*
B/X *nn* = 01..32, where *nn* is determined by the sequence of the CLUSTER-NODE state-
B/X ments during generation.

kc_con_str - LU6.1 connections

The data structure *kc_con_str* is defined for the object type KC_CON. In the case of KC_GET_OBJECT, UTM returns the properties and the current status of partner applications and connections for distributed processing via LU6.1 in *kc_con_str*.

Connections for distributed processing and their properties can be created and deleted dynamically (KC_CREATE_OBJECT object type KC_CON, KC_DELETE_OBJECT *subopcode1*=KC_IMMEDIATE or KC_DELAY, object type KC_CON, see also page 257).

Data structure kc_con_str

```
char co_name[8];
char pronam[8];
char bcamappl[8];
char lpap[8];
char termn[2];
```
X/W `char listener_port[5];`
X/W `char t_prot;`
X/W `char tsel_format;`
```
char state;
char auto_connect;
char connect_mode;
char contime_min[10];
char letters[10];
char conbad[5];
char ip_addr[15];
char co_deleted;
char ip_addr_v6[39];
char ip_v[2];
```

The fields in the data structure have the following meanings:

co_name
: Contains the name of the partner application that will be communicated with via the logical connection. The name is up to 8 characters long.

pronam
: Contains the name of the computer on which the partner application *co_name* is located. The name is up to 8 characters long.

B B B
: In a UTM application on BS2000 systems it is either the name of a Unix, Linux or Windows system, a BS2000 host or the TRANSDATA name of a communication computer for a heterogeneous link. *pronam* always contains data.

X/W X/W X/W
: In a UTM application on Unix, Linux or Windows systems, *pronam* contains the name of the partner computer that UTM uses to search for the IP address of the partner in the Name Service.

bcamappl
: Contains the name of the local application via which the connection to the partner application will be established. *bcamappl* can be the application name defined in the KDCDEF control statement MAX (APPLINAME) or a BCAMAPPL name of the application. The name is a maximum of 8 characters long. In order to be able to establish connections using this name, the local transport system must be known.

lpap
: Specifies the partner application to which the logical connection will be established. The name of the LPAP partner via which the partner application connects is specified.

termn
: Contains the code for the type of communication partner. The code is entered in the communication area header for the job-receiving services, i.e. for services in the local application that are started by a partner application. The code is defined by the user and serves to arrange the communication partners in groups of certain types. It is not evaluated by UTM.
The terminal code is two characters long.

X/W X/W listener_port
: Contains the port number of the transport address of the partner application.

X/W
: Possible values of *listener_port*: ´0´, ´102´, ´1025´ to ´32767´

X/W X/W
: If *listener_port* = ´0´, then no port number was specified when the CON object was created.

X/W X/W t_prot
: *t_prot* contains the address format with which the partner application signs on to the transport system. The address formats are specified as follows:

X/W X/W
: ´R´ RFC1006, ISO transport protocol class 0 using TCP/IP and the convergence protocol RFC1006.

X/W X/W X/W tsel_format
Contains the format indicator of the T-selectors of the partner address generated by the TNS generation tool.

X/W ´T´ TRANSDATA format

X/W ´E´ EBCDIC character format

X/W ´A´ ASCII character format

X/W X/W The meanings of the address formats are described in the "PCMX documentation" on page 20.

state Specifies the status of the partner application or its LPAP partner:

´Y´ The partner application is not disabled (ON). The connection to the partner application is or can be established.

´N´ The partner application is disabled (OFF). No logical connection to the partner application can be created.

The lock must be explicitly removed by the administration in order for the application to be able to work with the partner application (see *kc_lpap_str.state* on page 480).

auto_connect
Specifies if the connection to the partner application is automatically established at the start of the application:

´Y´ When the application is started, UTM will attempt to establish the connection automatically, i.e. if the partner application is available when the local application is started, then the connection is established after starting.

´N´ No automatic connection when starting.

connect_mode
Specifies the current status of the connection:

´Y´ The connection is established.

´W´ UTM is now attempting to establish the connection (waiting for connection)

´N´ The connection is not established.

contime_min
Specifies how many minutes the connection to the partner application has existed until now.

letters Contains the number of input and output messages for the partner application since the last start of the local application.

conbad
: Specifies how often the connection has been lost since the last start of the local application.

ip_addr
: Returns the IP address used by UTM for this connection from the object table of the application if the address is an IPv4 address.

B : BS2000 systems: openUTM always returns blanks in the *ip_addr* field.

X/W : An IPv6 address is returned in the *ip_addr_v6* field (see below).

X/W : UTM uses the address to set up connections to partner applications. UTM reads the IP address from the name service when the application is started using the generated processor name (*pronam*).

X/W : If the object tables do not contain an IPv4 address for the partner computer, UTM will return blanks in *ip_addr*.

co_deleted
: Indicates whether the transport connection was deleted from the configuration dynamically:

  ´Y´ The transport connection is deleted.

  ´N´ The transport connection is not deleted.

ip_addr_v6
: Returns the IP address used by UTM for this connection from the object table of the application if the address is an IPv6 address or an IPv4 address embedded in IPv6 format.

B : BS2000 systems: openUTM always returns blanks in the *ip_addr_v6* field.

X/W : An IPv4 address is returned in the *ip_addr* field (see above).

X/W : UTM uses the address in order to establish connections to the partner application. UTM reads the IP address from the Name Service using the generated computer name (*pronam*) when the application is started.
If there is no IPv6 address in the object tables for the partner computer, UTM returns blanks in *ip_addr_v6*.

ip_v
: Specifies whether the IP address used by UTM for this connection is an IPv4 or an IPv6 address:

X/W : 'V4' IPv4 Address.

X/W : 'V6' IPv6 address or IPv4 address embedded in IPv6 format.

B : BS2000 systems: openUTM always returns blanks in the *ip_v* field.

kc_db_info_str - Output database information

The data structure *kc_db_info_str* is defined for the object type KC_DB_INFO. If KC_GET_OBJECT is specified then UTM returns information on the generated database connections in *kc_db_info_str*.

Database connections are generated with the KDCDEF control statement DATABASE (BS2000 systems) or RMXA (Unix, Linux and Windows systems).

Data structure kc_db_info_str
`char db_id[2]`
`char db_type[8]`
`char db_entry_name[8]`
`char db_lib_info[54]`
`char db_xaswitch[54]`

The fields in the data structure have the following meanings:

db_id
: Specifies the ID of the database. The ID is a digit between 1 and 8 which represents the databases in the order in which they were generated. The ID is assigned internally by openUTM.

db_type
: Specifies the type of database system:

B ´UDS´
B ´LEASY´
B ´SESAM´
B ´CIS´
B ´DB´
´XA´

db_entry_name
: B In the case of a BS2000 database, the entry name of the BS2000 database is output. In the case of a BS2000 database with *db_type*=XA, the name of the XA switch generated with the ENTRY operand of the DATABASE statement is returned in *db_entry_name*. In a BS2000 system, this XA switch name is also returned in the *db_xaswitch* field.

X/W In the case of Unix, Linux and Windows systems, *db_entry_name* does not contain any relevant information.

db_lib_info

The meaning of this field is platform-specific.

B – On BS2000 systems, this field corresponds exactly to the LIB field in the KDCDEF statement DATABASE, i.e. it outputs information on the library from which the connection module to the database system was dynamically loaded. The field contains either the name of an object module library itself or a LOGICAL-ID as used during IMON installation in the format "LOGICAL-ID(logical-id)".

X/W – On Unix, Linux and Windows systems, this field contains the internal name of the loaded XA switch, e.g. "Oracle_XA".

db_xaswitch

The meaning of this field is platform-specific.

B – On BS2000 systems, the content of *db_entry_name* is returned in *db_xaswitch*.

X/W – On Unix, Linux and Windows systems, this field contains the name of the Resource Manager's XA switch. This name is defined in the XASWITCH parameter of the KDCDEF statement RMXA.

kc_edit_str - EDIT profile options (BS2000 systems)

The data structure *kc_edit_str* is defined for the object type KC_EDIT. With KC_GET_OBJECT, UTM returns information on EDIT profiles in *kc_edit_str*.

EDIT profiles are generated with the KDCDEF control statement EDIT. Screen functions and properties of the screen output in line mode are summarized in EDIT profiles. Each EDIT profile is assigned a name in the KDCDEF generation via which the corresponding set of edit options can be accessed from a program unit run.

A complete description of the edit options discussed in the following can be found in the TRANSDATA TIAM User Guide. You will find more detailed information on working with EDIT profiles in the openUTM manual „Programming Applications with KDCS”.

Data structure kc_edit_str

```
char ed_name[8];
char edit_mode;
char edit_bell;
char hcopy;
char hom;
char ihdr;
char locin;
char low;
char nolog;
char ohdr;
char saml;
char specin;
char ccsname[8];
```

The fields in the data structure have the following meanings:

ed_name
Contains the name of the EDIT profile whose edit options UTM will return. It is an alphanumeric name up to seven characters long.

edit_mode
Specifies the mode in which the messages will be output:

´E´ (extended line mode)
The messages are output in "extended line mode".

´I´ (info)
The message can be indicated in a special information line (system line) without important data being overwritten at the terminal.

´L´ (line mode)
The message is output in line mode. It can be structured using logical control characters. The message is prepared by the system.

´P´ (physical mode)
The message is physically input or output, i.e. without being prepared by the system.

´T´ (transparent mode)
The output message is transmitted transparently.

edit_bell
Specifies if an acoustic alarm is to be triggered when the message is output on the terminal. The contents of the field mean:

´Y´ An acoustic alarm will be triggered.

´N´ An acoustic alarm will not be triggered.

hcopy (**h**ard **copy**)
Specifies if an output message is also to be logged by a hardcopy printer connected to the terminal in addition to the output on the terminal.

´Y´ Logging of output messages on a hard-copy printer

´N´ No logging

hom (**hom**ogeneous)
Specifies if the output message is output without structure, i.e. homogeneously output.

´Y´ The message will be without structure

´N´ The message will be structured. In this case, a logical line is considered to be the unit of output.

ihdr (**i**nput **h**ea**d**e**r**)
Specifies if the message header of the input message is to be passed to the program unit.

´Y´ The message header of the input message will be passed.

´N´ The message header will not be passed.

locin (**loc**al **in**put parameter)
Specifies if local attributes in the input message are passed to the user as logical control characters.

´Y´ Local attributes in the input message are passed as logical control characters.

´N´ Local attributes are removed and not passed.

low (**low**er case)
Specifies if the input message passed to the program unit may also contain lowercase letters.

´Y´ Lowercase letters in the input message are passed to the program unit.

´N´ Lowercase letters are converted to uppercase before being passed to the program unit.

nolog (**no log**ical characters)
Specifies how non-printable characters will be handled by the system.

´Y´ The logical control characters will not be evaluated. All characters that are smaller than X'40' in the EBCDIC code will be replaced by substitute characters (SUB). Only printable characters will be allowed through.

´N´ All logical control characters are evaluated. Special physical control characters will be allowed through. Other characters smaller than X'40' will be replaced by substitute characters (SUB). Printable characters will be allowed through.

ohdr (**o**utput **h**ea**d**e**r**)
Specifies if the output message contains a message header. The length of the message header +1 will be entered in binary in the first byte of the message.

´Y´ The output message contains a message header.

´N´ The output message does not contain a message header.

saml (**sam**e **l**ine)
Specifies if a line feed at the beginning of the message is to be suppressed. The contents of *saml* is only significant for printers. The contents of the *saml* field have the following meaning:

´Y´ **No** line feed is executed at the beginning of the message.

´N´ The message starts at the beginning of the next line.

specin (**spec**ial **in**put)
Specifies which special options the edit profile contains for the input.

´C´ (confidential)
The input data is darkened when displayed on the terminal.

´I´ (ID card)
The next entry will be input via the ID reader.

´N´ (normal)
Normal input from the terminal.

ccsname (**c**oded **c**haracter **s**et **name**)
Contains the name of the character set (CCS name) used to prepare a message (see also the XHCS User Guide).

kc_gssb_str - Global secondary storage areas of the application

The data structure *kc_gssb_str* is defined for the object type KC_GSSB. With KC_GET_OBJECT, UTM returns the names of the global secondary storage areas (GSSB) currently exeisting in the application in *kc_gssb_str*. A global secondary storage area is used by KDCS program units for passing data between services.

Data structure kc_gssb_str
`char gs_name[8];`

The field has the following meaning:

gs_name
: Contains the name (up to 8 characters long) of the global secondary storage area.

kc_kset_str - Key sets of the application

The data structure *kc_kset_str* is defined for the object type KC_KSET. With KC_GET_OBJECT, UTM returns information on a key set in *kc_kset_str*.

The key or access codes of the application that were defined for data access control are grouped together in a logical key set.

You can assign a key set to a user, an LTERM partner, an LTERM pool, an (OSI-)LPAP partner or an access list. This controls access to TAC objects, for example. In this manner, the key set and the access privileges associated with it are made available to the clients or the partner application after establishing the logical connection or to the user after signing on to the application (see also the openUTM manual “Concepts und Functions”).

The key sets can be created with KC_CREATE_OBJECT, deleted with LKC_DELETE_OBJECT, or dynamically modified with KC_MOFDIFY_OBJECT. Which key set is assigned to a client, a partner application or a user is returned in the data structure of the object in the *kset* field.

KDCDEF implicitly creates the KDCAPLKS key set, which already contains all key codes.

mod[1]	**Data structure kc_kset_str**
–	`char ks_name[8];`
–	`char master;`
x(GPD)	`char keys[4000];`
–	`char ks_deleted;`

[1] Field contents can be modified with KC_MODIFY_OBJECT, see page 323f

The fields in the data structure have the following meanings:

ks_name
: Contains the up to 8 characters long name of the key sets. It is specified in KSET when the key set is created with KC_CREATE_OBJECT object type KC_KSET or at KDCDEF generation (KSET statement).

master
: Specifies if the key set is a master key set. A master key set contains all key or access codes needed to access the objects of the application, i.e. all key codes between 1 and the maximum specified in the KDCDEF generation in MAX KEYVALUE.

  ´Y´ The key set is a master key set.

  ´N´ The key set is not a master key set (default).

keys Specifies the key or access codes that belong to the key set.

A key or access code is an integer between 1 and the KEYVALUE set during the KDCDEF generation in the MAX statement. KEYVALUE is the largest possible key or access code of the application. KEYVALUE can lie between 1 and 4000.

keys consists of 4000 field elements, *keys*[0] to *keys*[3999]. The contents of the field elements are interpreted as follows:

keys[0] =
´0´: The key/access code 1 does not belong to this key set.
´1´: The key/access code 1 belongs to this key set.

keys[n] =
´0´: The key/access code n+1 does not belong to this key set.
´1´: The key/access code n+1 belongs to this key set.

keys[3999] =
´0´: The key/access code 4000 does not belong to this key set.
´1´: The key/access code 4000 belongs to this key set.

ks_deleted
Indicates whether the key set was deleted from the configuration dynamically:

´Y´ The key set is deleted.

´N´ The key set is not deleted.

kc_load_module_str - Load modules (BS2000 systems) or shared objects/DLLs (Unix, Linux and Windows systems)

The data structure *kc_load_module_str* is defined for the object type KC_LOAD_MODULE. In the case of KC_GET_OBJECT, UTM returns the following in *kc_load_module_str*:

B
B
- Information on the load modules that were generated with the KDCDEF control statement LOAD-MODULE.

X/W
X/W
- Information on the shared objects or DLLs that were generated with the KDCDEF control statement SHARED-OBJECT.

Load modules and shared objects/DLLs must be statically generated.

Using a KDCADMI call with the operation code KC_MODIFY_OBJECT and the object type KC_LOAD_MODULE, you can replace individual load modules or shared objects or DLLs during the application run.

mod[1]	Data structure kc_load_module_str
–	`char lm_name[32];`
x(GID)	`char version[24];`
–	`char lib[54];`
–	`char load_mode;`
–	`char poolname[50];`
–	`char version_prev[24];`
–	`char changeable;`
–	`char change_necessary;`
–	`char altlib;`

[1] The contents of the field can be modified using KC_MODIFY_OBJECT; see page 323f

The fields in the data structure have the following meanings:

lm_name
: Contains the name of the load module or shared object or DLL. The name can be up to 32 characters long.

version
: UTM returns the version number of the load module, shared object or DLL currently loaded in *version*. *version* can be a maximum of 24 characters long.

lib The contents of *lib* have the following meaning:

B B – In UTM applications on BS2000 systems, UTM returns the program library from which the load module will be loaded in *lib*.

X/W X/W – In UTM applications running on Unix, Linux or Windows systems, *lib* contains the directory in which the shared object /DLL is stored.

The name in *lib* can be up to 54 characters long.

load_mode

Contains the load mode of the load module, shared object or DLL. The load mode determines when and to where a load module/shared object/DLL will be loaded.

´U´ (STARTUP)
The load module or shared object/DLL is loaded as an independent unit at the start of the application.

B B B When a load module is loaded, external references from all modules of the UTM application that were already loaded, from all nonprivileged subsystems and from the class 4 storage are resolved.

´O´ (ONCALL)
The load module/shared object/DLL is loaded as an independent unit when one of its program units or VORGANG exits are called for the first time.

B B B When a load module is loaded, external references from all modules of the UTM application that were already loaded, from all non-privileged subsystems and from the class 4 storage are resolved.

B B B B If several processes are utilized at one time, then this load module must not be overwritten in the library (LIB=...) during the application run. Otherwise, different states of the load module may perhaps be executed in an application run.

B B B ´S´ (STATIC)
The load module is statically bound in the application program. The load module cannot be replaced during an application run.

B B B B ´P´ (POOL)
The load module is loaded into a common memory pool (see *poolname*) at the start of the application. The load module consists only of one public slice (no private slice).

B B B B B ´T´ (POOL/ STARTUP)
The public slice of the load module is loaded into a common memory pool (see *poolname*) at the start of the application. The private slice belonging to the load module is loaded into the local process memory after that (private slice with load mode STARTUP).

´C´ (POOL/ONCALL)
The public slice of the load module is loaded into a common memory pool (see *poolname*) at the start of the application. The private slice belonging to the load module is loaded into the local process memory when the first program unit assigned to this load module is called (private slice with load mode ONCALL).

poolname
poolname is only specified if the load module or its public slice will be loaded into a common memory pool (*load_mode*=´P´, ´T´ or ´C´). *poolname* then contains the name of the common memory pool. The name can be up to 50 characters long.

version_prev
Contains the previous version of the load module/shared object/DLL, i.e. the version that was loaded before the last program change.
If the load module/shared object/DLL has not yet been replaced or is not replaceable, then *version_prev* contains blanks.

changeable
Specifies if the load module/shared object/DLL can be replaced.

´Y´ The load module/shared object/DLL can be replaced during the application run.

´N´ The load module/shared object/DLL cannot be replaced during the application run.

change_necessary
change_necessary is only relevant for load modules that either lie completely within a common memory pool or whose public slice lies in common memory pool. *change_necessary* specifies if this load module has been marked for a program change.

Load modules in the common memory pool must then be marked for a program change with KC_MODIFY_OBJECT. The actual replacement must then be executed with KC_CHANGE_APPLICATION.

´Y´ The load module is marked for replacement. A program change using KC_CHANGE-APPLICATION is necessary to replace the load module.

´N´ The load module is not marked for replacement.

altlib Specifies if the load module will be loaded with the BLS autolink function.

´Y´ Load with autolink

´N´ Load without autolink

kc_lpap_str - Properties of LU6.1 partner applications

The data structure *kc_lpap_str* is defined for the object type KC_LPAP. In the case of KC_GET_OBJECT, UTM returns the properties of an LPAP partner in *kc_lpap_str*.

An LPAP partner is a logical connection point for an LU6.1 partner application. LPAP partners are defined during the static generation with KDCDEF and are assigned to the LU6.1 partner applications. You can make the assignment to a real partner application at generation or dynamically when creating a new CON object.

	mod[1]	Data structure kc_lpap_str
	–	`char lp_name[8];`
	–	`char kset[8];`
	–	`char lnetname[8];`
B	–	`char netprio;`
	–	`char permit;`
	–	`char qlev[5];`
	–	`char rnetname[8];`
	x(GPD)	`char state;`
	x(GPD)	`char auto_connect;`
	–	`char contwin;`
	–	`char dpn[8];`
	x(GPD)	`char idletime_sec[5];`
X/W	–	`char map;`
	–	`char paccnt[2];`
	–	`char plu;`
	x(A)	`char connect_mode;`
	x(IR)	`char quiet_connect;`
	x(IR)	`char bcam_trace;`
	–	`char out_queue[5];`
	–	`char nbr_dputs[10];`
	–	`char master[8];`
	–	`char bundle;`
	–	`char out_queue_ex[10];`

[1] The contents of the field can be modified using KC_MODIFY_OBJECT; see page 325f

The fields in the data structure have the following meanings:

lp_name
: Contains the name of the LPAP partners, i.e. the logical name of the partner application. Through this name the local application initiates communication with the partner application. *lp_name* only has meaning in the local application.

kset
: Contains the name of the key set that is assigned to the partner application. The key set specifies the access privileges of the partner application within the local application, meaning that the partner application may only use the transaction codes that are either secured by a lock code for which the key set contains the appropriate key or access code or that are not secured by a lock code.

lnetname
: *lnetname* is only relevant for heterogeneous links.

  lnetname contains the name of the local UTM application under which the local application is known in the partner application.

B netprio
B : Contains the transport priority used in the transport connection assigned to this LPAP partner.

B ´M´
: "Medium" transport priority

B ´L´
: "Low" transport priority

permit
: Specifies the privileges that the partner application has within the local application.

  ´A´ (ADMIN)
  : The partner application has administration privileges, it may execute all administration functions in the local application.

  ´N´ (NONE)
  : The partner application does not have any administration privileges.

  B B B
  : If the local application is a UTM application on a BS2000 system, then the partner application is also not allowed to execute any UTM SAT administration functions.

  B B B ´B´ (BOTH)
  : The partner application may execute administration functions as well as UTM SAT administration functions in the local application.

  B B B B ´S´ (SAT)
  : The partner application has UTM SAT administration privileges. It may execute preselection functions in the local application, i.e. it can enable or disable the SAT logging for certain events.

qlev (**q**ueue **lev**el)
qlev specifies the maximum number of asynchronous messages that may be in the local message queue for the partner application. If this control value is exceeded, then any additional asynchronous jobs sent to the partner application will be rejected (i.e. ´40Z´ will be returned for any APRO-AM calls thereafter).

rnetname
rnetname is only relevant for heterogeneous links.

rnetname contains the VTAM name of the partner CICS application or IMS application.

state Contains the status of the LPAP partner:

´Y´ The LPAP partner is not disabled. A connection to the partner application can be established or there already is an established connection.

´N´ The LPAP partner is disabled. No connections to the partner application can be established.

auto_connect
Specifies if the connection to the partner application is automatically established when the local application is started:

´N´ The connection is not automatically established; it must be established by the administrator.

´Y´ When the local application is started, UTM will automatically establish the connection to the partner application as long as the partner application is available at that time.

If automatic connecting is defined in both applications (local application and partner application), then the connection between the two is automatically established as soon as both applications are available.

contwin (**cont**ention **win**ner)
Specifies if the partner application is the contention winner in the session connecting the local application and the partner application. The contention winner administers the session and controls how resources are allocated for jobs in the session.

´Y´ The partner application is the contention winner.

´N´ The local application is the contention winner.

In any case, jobs from the local application as well as from the partner application may be started. In case of a conflict, such as when the local and the partner application want to start a job at the same time, the job from the contention winner will be started in the session.

dpn (**d**estination **p**rocess **n**ame)
dpn is only meaningful for connections to IBM systems.

dpn contains the name of the instance that processes asynchronous messages.

idletime_sec
Contains the maximum time in seconds that a session to the partner application may be in the idle state before UTM closes the connection to the partner application. The idle state means that the session is not handling any jobs.

idletime_sec = ´0´ means that the idle state will not be monitored.

Minimum value: ´60´
Maximum value: ´32767´

X/W map Specifies whether or not UTM executes any message handling when exchanging X/W messages without any formatting flags with the partner application.

X/W ´U´ (USER)
X/W UTM does not execute any message handling, i.e. the data in the message
X/W is transmitted unchanged to the partner application.

X/W ´S´ (SYSTEM)
X/W UTM converts the messages to the partner application from ASCII to
X/W EBCDIC before sending. Messages received from the partner application
X/W are converted from EBCDIC to ASCII. The message in these cases must
X/W contain printable characters only.

X/W UTM does not execute any default message handling for formatted messages.

paccnt (**pac**ing **c**ou**nt**)
Contains the number of parts of a long message that the local application may receive without having to acknowledge.

A pacing value in *paccnt* that is too large can lead to bottlenecks in the network. If *paccnt* = ´0´, there is no limit to the number of parts of a message that can be received before acknowledging.

plu (**p**rimary **l**ogical **u**nit)
Specifies if the partner application is responsible for establishing the session, i.e. if the partner application is the ’primary logical unit’ (PLU).

´Y´ The partner application is the ’primary logical unit’.

´N´ The local application is the ’primary logical unit’.

connect_mode

Specifies the status of the connection to the partner application.

´Y´ The partner application is currently connected to the application.

´N´ The partner application is not currently connected to the application.

´W´ UTM is currently attempting to establish a connection to the partner application (WAIT).

quiet_connect

Specifies if the QUIET property is set for the connection to the LPAP partner. QUIET means that UTM closes the connection to the partner application as soon as the sessions generated for the partner application do not contain any more jobs. No more new dialog jobs are accepted for the partner application.

´Y´ The QUIET property is set.

´N´ The QUIET property is not set.

bcam_trace

Specifies whether the BCAM trace is explicitly enabled or disabled for the LPAP partner of the partner application. The trace function that monitors connection-specific activity within a UTM application (for example, the BCAM trace function on BS2000 systems) is called the BCAM trace. The BCAM trace can be enabled for all connections of the application (i.e. for all LPAP and LTERM partners) or explicitly for certain LTERM or LPAP partners.

´Y´ The BCAM trace was explicitly enabled for this LPAP partner.
If the BCAM trace was enabled for all connections of the UTM application, then ´N´ will be returned in *bcam_trace*.
You can determine if the BCAM trace is enabled for all connections by, for example, calling KC_GET_OBJECT with the KC_DIAG_AND_ACCOUNT_PAR parameter type. Then *bcam_trace*=´Y´ will be returned in *kc_diag_and_account_par_str*.

´N´ The BCAM trace was not explicitly enabled for this LPAP partner.

You can enable or disable the BCAM trace during the application run.

out_queue

The number of messages currently being stored temporarily in the local message queue of the partner application and which must still be sent to the partner application.

If the number of messages is greater than 99999, then the number is not displayed in full. You should therefore use the field *out_queue_ex* since larger numbers can be entered in full here.

nbr_dputs
: The number of pending time-driven jobs for this LPAP whose starting time has not yet been reached.

master
: If the LPAP partner is a slave in an LU6.1 LPAP bundle then the master LPAP partner of the bundle is returned in *master*.

bundle
: Specifies whether the LPAP partner belongs to an LPAP bundle.

  ´N´ The LPAP partner does not belong to an LPAP bundle.

  ´M´ The LPAP partner is the master of an LPAP bundle.

  ´S´ The LPAP partner is a slave in an LPAP bundle.

out_queue_ex
: see *out_queue* on page 482.

kc_lses_str - LU6.1 sessions

The data structure *kc_lses_str* is defined for the object type KC_LSES. In the case of KC_GET_OBJECT, UTM returns the properties of sessions to LU6.1 partners of the application in *kc_lses_str*.

Sessions to LU6.1 partners can be dynamically created with KC_CREATE_OBJECT, deleted with KC_DELETE_OBJECT, or modified with KC_MODIFY_OBJECT.

A session is identified using the name specified in the LSES statement.

mod[1]	Data structure kc_lses_str
–	`char ls_name[8];`
–	`char lpap[8];`
–	`char rses[8];`
–	`char con[8];`
–	`char pronam[8];`
–	`char bcamappl[8];`
x(A)	`char connect_mode;`
x(IR)	`char quiet_connect;`
–	`char lses_user[8];`
–	`char ls_deleted;`
–	`char ls_used;`
–	`char ptc;`
–	`char node_name[8];`

[1] The contents of the field can be modified using KC_MODIFY_OBJECT; see page 328f

The fields in the data structure have the following meanings:

ls_name (**l**ocale **s**ession **name**)
: Contains the name of the session within the local application (local half-session name).

lpap
: Specifies to which partner application the session is assigned. *lpap* contains the name of the LPAP partner via which the partner application is connected.

rses
: (**r**emote **ses**sion name)
Contains the name that the session has in the partner application (remote half-session name).

con, pronam, bcamappl

These parameters uniquely identify the transport connection that has been or will be established for this session.

con
Contains the name of the transport connection to the partner application defined at dynamic creation (KC_CREATE_OBJECT object type KC_CON) or during the KDCDEF generation in the CON statement.

pronam
The name of the computer on which the partner application is running.

bcamappl
Contains the name of the local UTM application (BCAMAPPL name) via which the connection to the partner application will be established.

connect_mode

Specifies if a transport connection is established for the session.

´Y´ A transport connection to the partner application is established for the session.

´N´ No transport connection is established for the session at the present time.

quiet_connect

Specifies if the QUIET property is set for the connection. QUIET means that UTM closes the connection as soon as the session contains no more jobs. No more new dialog jobs are accepted for the partner application.

´Y´ The QUIET property is set.

´N´ The QUIET property is not set.

lses_user

Name of the job submitter currently using the session. *lses_user* specifies who started the job-submitting service.
If the job-submitting service is running in the local application for a dialog job, then the user ID or LTERM partner of the client that started the service is specified in *lses_user*.

If the job-receiving service is running in the local application for a dialog job, i.e. the local application is processing the job, then the local session name (*ls_name*) is output in *lses_user*.

If asynchronous messages are transmitted in the session, then the local session name (*ls_name*) is output in *lses_user* in this case, too.

ls_deleted
: Indicates whether the LSES object was deleted from the configuration dynamically.

  ´Y´ The session is deleted.

  ´N´ The session is not deleted.

ls_used
: Indicates whether or not the session is being used.

  ´Y´ The session is being used.

  ´N´ The session is not being used.

ptc
: Indicates the state of the session.

  ´Y´ The session is in the PTC state (prepare to commit).

  ´N´ The session is not in the PTC state.

node_name
: Only in UTM cluster applications: Reference name of the node application to which the session is assigned.

kc_ltac_str - Transaction codes of remote services (LTAC)

The data structure *kc_ltac_str* is defined for the object type KC_LTAC. In the case of KC_GET_OBJECT, UTM returns the properties of transaction codes that are defined in the local application for remote service programs in *kc_ltac_str*.

LTAC objects can be dynamically created with KC_CREATE_OBJECT, deleted with KC_DELETE_OBJECT, or modified with KC_MODIFY_OBJECT.

mod[1]	Data structure kc_ltac_str
–	`char lc_name[8];`
–	`char lpap[8];`
–	`union kc_rtac rtac;`
–	`char rtac_lth[2];`
–	`char code_type;`
x(GPR)	`char state;`
x(GPR)	`char accesswait_sec[5];`
x(GPR)	`char replywait_sec[5];`
–	`char lock_code[4];`
–	`char ltac_type;`
–	`char ltacunit[4];`
–	`char used[10];`
–	`char access_list[8];`
–	`char deleted;`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 330f

The fields in the data structure have the following meanings:

lc_name
: Contains the local transaction code that was defined for the remote service program (LTAC name).

lpap
: Specifies to which partner application the service program belongs. *lpap* contains the name of the LPAP partner assigned to the partner application.

rtac (**r**emote **tac**)
Name of the transaction code for the service or service program in the partner application (recipient TPSU title; RTAC name). The name can be a string or a number. The name is returned in the following data structure:

union kc_rtac
`char name64[64];`
`char name8[8];`

The field of the union in which the RTAC name is stored depends on the code type assigned to the RTAC name. The code type is returned in the *code_type* field.

rtac_lth
Specifies how long the name (recipient TPSU title) returned in *rtac* is. The number of bytes used is specified in *rtac*.

Minimum value: ´1´
Maximum value: ´64´

code_type
Specifies which code type will be used internally by UTM for the RTAC name. Based on the *code_type* you can determine in which field of the union the RTAC name is stored.

´I´ (INTEGER)
The TAC name in *rtac* is a positive integer between 0 and 67108863.

The RTAC name will be returned in the *name8* field of the *rtac* union (the first 8 bytes of the union are right-justified).

RTAC names of code type INTEGER are only permitted for partner applications that are not UTM applications and that communicate using the OSI TP protocol.

´P´ (PRINTABLE-STRING)
The TAC name in *rtac* is specified as a string. It is a maximum of 64 characters long. It is case sensitive.

A TAC name with the PRINTABLE-STRING code type can contain the following characters:
– A, B, C, . . . , Z
– a, b, c, . . . , z
– 0, 1, 2, . . . , 9
– the special characters - : ? = , + . ( ) / ␣ (blank)

The TAC name will be returned in the *name64* field of the *rtac* union. The elements in *kc_rtac.name64* after the length specified in *rtac_lth* are filled with blanks.

´T´ T61-STRING
rtac contains a T61 string. For the T61-STRING code type UTM supports all characters of the PRINTABLE-STRING code type in addition to the following special characters: $ > < & @ # % ; * _

The TAC name will be returned in the *name64* field of the *rtac* union. The elements in *kc_rtac.name64* after the length specified in *rtac_lth* are filled with blanks.

RTAC names for which the STANDARD code type was specified at dynamic creation or at KDCDEF generation are stored internally, depending on the characters used, as a PRINTABLE-STRING or a T61-STRING. For this reason, either PRINTABLE-STRING or T61-STRING is output for RTACs generated with ´S´ or STANDARD.

state Contains the status of the transaction codes in *lc_name*:

´Y´ The transaction code is not disabled. Jobs for the corresponding remote service will be accepted.

´N´ The transaction code is disabled. Jobs for the corresponding remote service will not be accepted.

accesswait_sec
The time to wait in seconds after a remote service (LTAC call) requests the appropriation of a session (possibly including the establishing of the connection) or the maximum time to wait for an association to be established.

A wait time *accesswait_sec* ≠ 0 for asynchronous jobs (LTAC with *ltac_type*=´A´) means that the job will always be placed in the local message queue for the partner application. Dialog jobs are accepted.

A wait time *accesswait_sec*=0 means:
Dialog TACs will be rejected if no session or association to the partner application is generated for which the local application is the "contention loser".

For asynchronous TACs, the asynchronous job will be rejected with a return code in the FPUT call if there is no logical connection to the partner application. If a logical connection to the partner application exists, then the message will be placed in the local message queue.

Dialog jobs are rejected, independent of the value in *accesswait_sec*, if no logical connection to the partner exists. At the same time, a connection shutdown is initiated.

Minimum value: ´0´
Maximum value: ´32767´

replywait_sec

The maximum time in seconds that UTM will wait for an response from a remote service. By limiting the wait time you can guarantee that the wait time for clients or users on the terminal will not be indefinite.

replywait_sec = ´0´ means: no wait time limit.

Minimum value: ´0´
Maximum value: ´32767´

lock_code

Contains the lock code assigned to the remote service within the local application (access protection). *lock_code* can contain a number between ´0´ and ´4000´.
In KC_CREATE_OBJECT, the maximum value that can be contained by *lock_code* is the maximum value defined using the KEYVALUE operand of the KDCDEF statement MAX. ´0´ means that the LTAC is not protected by a lock code.

When an LTAC object is created, only *lock_code* or *access_list* can be specified (see below). If you modify an LTAC object, you can change the current value or remove the lock code by specifying ´0´.

If neither *lock_code* nor *access_list* is defined, *lc_name* is not protected and every user of the local application can start the remote service program.

ltac_type

Specifies if the local application with the remote service jobs processes in the dialog or if asynchronous jobs will be passed to the partner service.

´D´ Jobs sent to the partner service are processed in dialog mode.

´A´ The partner service is started asynchronously (by means of message queuing).

used Contains the number of jobs sent to the remote service since the start of the local application. *used* also specifies how often the LTAC has been called within the current application run.

The counter is reset to 0 every time the application is started.

ltacunit

Contains the number of accounting units charged for each *ltac* call in the accounting phase of UTM Accounting. The accounting units are added to the accounting unit counter of the user ID that called *ltac*.

For more information on accounting see also the openUTM manual “Generating Applications” and openUTM manual “Using openUTM Applications”.

access_list

Can contain the name of a key set that describes the access rights of users who are permissible for *lc_name*.

When an LTAC object is created, only *lock_code* or *access_list* can be specified (see above). When you modify an LTAC object, you can change the current entry or remove the key set by specifying 8 blanks.

If neither *lock_code* nor *access_list* is defined, *lc_name* is not protected and any user of the local UTM application can start the remote service program.

deleted

Indicates whether *lc_name* was deleted from the configuration dynamically.

´Y´ *lc_name* is deleted.

´N´ *lc_name* is not deleted.

kc_lterm_str - LTERM partners

The data structure *kc_lterm_str* is defined for the object type KC_LTERM. In the case of KC_GET_OBJECT, UTM returns the properties of LTERM partners and specifies to which client or printer the LTERM partner is presently assigned in *kc_lterm_str*.

LTERM partners can be dynamically created with KC_CREATE_OBJECT, deleted with KC_DELETE_OBJECT or modified with KC_MODIFY_OBJECT.

	mod[1]	Data structure kc_lterm_str
	–	`char lt_name[8];`
	–	`char kset[8];`
B	–	`char locale_lang_id[2];`
B	–	`char locale_terr_id[2];`
B	–	`char locale_ccsname[8];`
	–	`char lock_code[4];`
	x(GPD)	`char state;`
	–	`char usage_type;`
	–	`char user_gen[8];`
	–	`char cterm[8];`
B	x(GPD)[2]	`char format_attr;`
B	x(GPD)[2]	`char format_name[7];`
	–	`char plev[5];`
	–	`char qamsg;`
	–	`char qlev[5];`
	–	`char restart;`
B	–	`char annoamsg;`
B	–	`char netprio;`
	x(PD)	`char master[8];`
	–	`char pterm[8];`
	–	`char pronam[8];`
	–	`char bcamappl[8];`
	–	`char user_curr[8];`
	x(A)	`char connect_mode;`
	x(IR)	`char bcam_trace;`
	–	`char bundle;`
	–	`char pool;`

mod[1]	Data structure kc_lterm_str
–	`char out_queue[5];`
–	`char incounter[10];`
–	`char seccounter[5];`
–	`char deleted;`
–	`char nbr_dputs[10];`
–	`char lt_group;`
–	`char out_queue_ex[10];`
B –	`char kerberos_dialog`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 332f

[2] When changing the start format with KC_MODIFY_OBJECT you must always enter data in *format_name* and *format_attr*.

The fields in the data structure have the following meanings:

lt_name
: Name of the LTERM partner; It can also be an LTERM partner that belongs to an LTERM pool.

  The program units of the application communicate with the clients, printers and TS applications (no server-server communication) that are assigned to the LTERM partner using this name.

kset
: Specifies which key set is assigned to this LTERM partner (access privileges). *kset* contains the name of the key set.

  The key set limits the access privileges of a client/user that connects via this LTERM partner. A client or client program can only start a service protected by a lock code or an access list when the key or access code corresponding to the lock code or the access list is contained both in the key set of the user ID under which the client or client program signs on and in the key set of the associated LTERM partner.

B locale_lang_id, locale_terr_id, locale_ccsname
: These contain the three components of the locale assigned to the partner. The locale defines the language environment of the client that is connected to the application via this LTERM partner. The language environment is relevant if messages and UTM messages of the application are to be output in different languages. The LTERM-specific language environment is set when outputting asynchronous messages and in the first part of the sign-on service if the user-specific environment has not been set.

B See the openUTM manual “Generating Applications” for more information on multilingual capabilities.

B B locale_lang_id
Contains the up to two characters long language code.

B B *locale_terr_id*
Contains an up to two characters long territory code.

B B B B *locale_ccsname*
(**c**oded **c**haracter **s**et **name**)
Contains the up to 8 characters long name of an extended character set (CCS name; see also the XHCS User Guide).

lock_code
Contains the lock code assigned to the LTERM partner (access protection). Only users/clients who possess the corresponding key code may connect via this LTERM partner.

The *lock_code* can contain a number between ´0´ and ´4000´.
In KC_CREATE_OBJECT, the maximum value that can be contained by *lock_code* is the maximum value defined using the KEYVALUE operand of the KDCDEF statement MAX.
´0´ means that the LTERM partner is not protected by a lock code.

state Specifies if the LTERM partner is currently disabled.

´Y´ The LTERM partner is not disabled.

´N´ The LTERM partner is disabled. No user/client can connect to the application at the present time via this LTERM partner.

usage_type
Type of LTERM partner

´D´ The LTERM partner is configured as a dialog partner. The client as well as the local application can send messages via the connections between the client (*pterm*) and local application.

´O´ The LTERM partner is configured for an output medium (printer). Messages can only be sent from the application to the client/printer (*pterm*).

user_gen
user_gen contains data if the LTERM partner was configured as a dialog partner (*usage_type* = ´D´).

If the LTERM partner is assigned with a terminal, *user_gen* contains the user ID for which UTM is to execute an automatic KDCSIGN (automatic sign-on) when the logical connection between the client and the application is established (defined in *kc_lterm_str.user*).

If the LTERM partner is assigned to a UPIC client or a TS application (*ptype*=´APPLI´ or ´SOCKET´), then *user_gen* contains the connection user ID.

cterm *cterm* only contains data if the LTERM partner (*usage_type* = ´O´) is assigned to a printer control LTERM. *cterm* then contains the name of the printer control LTERM, which can be a maximum of 8 characters long.

A printer control LTERM is assigned one or more printers. Asynchronous jobs in the message queues of the printers, the output of messages on the printers and the printer itself can be administered via the printer control LTERM (see chapter “Administering message queues and controlling printers” on page 799).

B B B format_attr, format_name
These parameters are only relevant when the LTERM partner is assigned to a terminal.

B B B *format_attr* and *format_name* define the LTERM-specific start format. An LTERM-specific start format is only useful in applications without user IDs and in applications with their own sign-on service.

B B B In applications without user IDs, the start format will be output on the terminal after establishing the connection between the terminal and the application instead of the message K001 as long as no LTERM-specific restart is being executed.

B B In applications generated with user IDs, the name of the start format can be queried during the first part of the sign-on service (with SIGN ST).

B B *format_attr*
Contains the format code:

B B B B ´A´ (format attribute ATTR)
The start format is a format with user attributes. The properties of the format fields can be changed by the KDCS program unit. The format name at the KDCS program interface is +*format_name*.

B B B B ´N´ (format attribute NOATTR)
The start format is a format without user attributes. Neither field properties nor format properties can be changed by the KDCS program units. The format name at the KDCS program interface is **format_name*.

B B B B B ´E´ (format attribute EXTEND)
The start format is a format with expanded user attributes. The properties of the format fields as well as global format properties can be changed by the KDCS program unit. The format name at the KDCS program interface is #*format_name*.

B B B *format_name*
Contains the name of the start format. The name can be up to 7 characters long and contains only alphanumeric characters.

plev (**p**rint **lev**el)
If the LTERM partner is a dialog partner, then *plev*=´0´ is always returned.

If the LTERM partner is assigned to an output medium (printer), then *plev* contains the control value for the number of print jobs that are temporarily stored in the message queue of the LTERM partner. UTM collects the messages for the corresponding printer until the control value specified in *plev* is reached. Then UTM attempts to establish the connection to the printer. The connection is closed when no more messages for the printer are in the queue. The control value is specified when adding the LTERM partner to the configuration.

plev=´0´ means that no control value is defined and UTM can temporarily store any number of print jobs in the queue without having to close the connection to the printer.

qamsg (**q**ueue **a**synchronous **m**e**s**sa**g**e)
Specifies whether asynchronous jobs are temporarily stored in the message queue of the LTERM partner even if the client/printer of the LTERM partner is not connected to the application.

´Y´ An asynchronous job is placed in the message queue of the LTERM partner even if no connection to the client/printer exists.
qamsg=´Y´ is not possible for *restart*=´N´.

´N´ An asynchronous job sent to this LTERM partner is rejected (return codes KCRCCC=44Z and KCRCDC=K705) if the corresponding client/printer is not connected to the application.

qlev (**q**ueue **lev**el)
Contains the maximum number of asynchronous messages that UTM may temporarily store in the message queue of the LTERM partner at one time. If this control value is exceeded, openUTM rejects any further FPUT or DPUT calls for this LTERM partner with 40Z. The control value is specified when adding the LTERM partner to the configuration.

restart Only relevant if the LTERM partner is assigned to a client. *restart* specifies how UTM will handle asynchronous messages that are in the message queue of the LTERM partner when shutting down a connection to the client.

´Y´ Asynchronous messages to this client remain in the queue when a connection is shut down. If no user IDs (USER) were generated in the application, then UTM will execute an automatic restart for these LTERM partners.

´N´ UTM deletes all asynchronous messages that are temporarily stored in the message queue of the LTERM partner when a connection is shut down. If the job is a job complex, then a negative confirmation job is activated.

If no user IDs (USER) were generated in this application, then UTM will not execute an automatic restart for the LTERM partner.

B B annoamsg (**anno**unce **a**synchronous **m**e**s**sa**g**e)
Is only of relevance for LTERM partners assigned to a terminal.

B B *annoamsg* specifies if UTM will announce asynchronous messages on the terminal with a UTM message in the system line before outputting.

B B B ´Y´ UTM announces every asynchronous message to this terminal with the K012 UTM message in the system line. The user must then explicitly request the asynchronous message with the KDCOUT command.

B B ´N´ Asynchronous messages are output on the terminal immediately, i.e. without announcement. KDCOUT is not permitted.

B B netprio Specifies the transport priority used on the transport connection between the application and the client/printer.

B ´M´ "Medium" transport priority

B ´L´ "Low" transport priority

master

The meaning of this field varies according to the operation code. You can establish what type of LTERM is involved from the *lt_group* parameter.

KC_GET_OBJECT

– The associated master LTERM is returned here in the case of a slave LTERM of an LTERM bundle.

– The associated primary LTERM is returned here in the case of an alias LTERM of an LTERM group.

KC_MODIFY_OBJECT

– For connection bundles: Exchange of two master LTERMs. The LTERM specified in *master* must be the master of an LTERM bundle. The master is specified with which the slaves are to be exchanged.

  You can only use this functionality in standalone UTM applications.

– For LTERM groups: Reassignment of a group LTERM to a different LTERM group. The LTERM that you specify in *master* must either be a normal LTERM, a primary LTERM of an LTERM group or a master LTERM of an LTERM bundle.

  You can only use this functionality in standalone UTM applications.

  A normal LTERM must fulfill the following conditions:

  – A PTERM with the PTYPE APPLI or SOCKET must be assigned to the LTERM.

  – The LTERM must not be a slave LTERM of an LTERM bundle.

  – The LTERM must have been generated with USAGE=D.

  The primary LTERM of the group to which the LTERM is to be added is specified.
  If the LTERM specified in *master* is already a primary LTERM of an LTERM group, the LTERM specified for *lt_name* is added to its LTERM group.
  If the LTERM specified in *master* was not a primary LTERM, a new LTERM group is created. The LTERM specified in *lt_name* is added to the new LTERM group. Primary LTERM is the LTERM specified in *master*.

pterm Name of the client/printer (PTERM name) currently assigned to this LTERM partner. If the LTERM partner is not currently assigned to a client/printer, then *pterm* contains blanks. The assignments between the LTERM partner and the client/printer can be changed; see also *kc_pterm_str* on page 529.

pronam

Name of the computer on which the client can be found or to which the printer is connected.

If the LTERM partner is not currently assigned a client/printer, the field contains blanks.

B
B
B
B

In UTM applications on BS2000 systems *pronam* always contains data if the LTERM partner is assigned a client or printer. The name in *pronam* is identical to the name of the computer specified for the BCAM generation for this computer. If the LTERM partner is assigned to an RSO printer, then *pronam* contains the value ´*RSO´.

X/W
X/W

In UTM applications running on Unix, Linux or Windows systems, *pronam* contains blanks if the LTERM partner is assigned to a local client or printer.

bcamappl

Name of the local UTM application (BCAMAPPL name) via which the connection to the client/printer will be established.

If the LTERM partner is assigned to a terminal or printer, then *bcamappl* always contains the name of the application that was specified for the KDCDEF generation in MAX APPLINAME.
If the LTERM partner is assigned to a UPIC client or a TS application, then *bcamappl* contains the application name (BCAMAPPL name) assigned to the client when it was added.

user_curr

User ID of the user currently connected with the application through this LTERM partner. If there is currently no connection, *user_curr* is padded with blanks.

If a connection to a terminal is established, but no user is as yet signed on, *user_curr* is also padded with blanks.

If a connection to a UPIC client or to a TS application is established, we distinguish the following situations:

– The application is generated with SIGNON MULTI-SIGNON=YES (see *kc_signon_str.multi_signon* on page 646)
user-curr contains the connection user ID (*user_gen*) until a client signs on with a “true” user ID for which *kc_user_str.restart*=´Y´.

– The application is generated with SIGNON MULTI-SIGNON=NO.
user_curr contains the connection user ID (*user_gen*) until a client signs on with a “true” user ID.

connect_mode
: Specifies if the client or printer presently assigned to this LTERM partner is currently connected to the application.

  ´Y´ The client/printer is currently connected to the application.

  ´W´ UTM is currently attempting to establish a connection to the client/printer.

  ´N´ The client/printer is not currently connected to the application.

bcam_trace
: Specifies if the BCAM trace is explicitly enabled or disabled for this LTERM partner. The trace function that monitors connection-specific activity within a UTM application (for example the BCAM trace function on BS2000 systems) is called the BCAM trace. The BCAM trace can be enabled for all connections of the application (i.e. for all LPAP and LTERM partners) or explicitly for certain LTERM or LPAP partners.

  ´Y´ The BCAM trace was explicitly enabled for this LTERM partner.
  If the BCAM trace was enabled for all connections of the UTM application, then ´N´ will be returned in *bcam_trace*.

  You can determine if the BCAM trace is enabled for all connections by, for example, calling KC_GET_OBJECT with the KC_DIAG_AND_ACCOUNT_PAR parameter type. Then *bcam_trace*=´Y´ will be returned in *kc_diag_and_account_par_str*.

  ´N´ The BCAM trace was not explicitly enabled for this LTERM partner.

  You can enable or disable the BCAM trace during the application run.

bundle
: bundle is only relevant for LTERM partners that are assigned to a printer or an LTERM bundle. *bundle* specifies if the LTERM partner belongs to a printer pool or an LTERM bundle.

  ´Y´ The printer is assigned to a printer pool.

  ´N´ The printer is not assigned to a printer pool.

  'M' The LTERM partner is a master of an LTERM bundle.

  'S' The LTERM partner is a slave of an LTERM bundle.

pool
: Specifies if the LTERM partner belongs to an LTERM pool.

  ´Y´ The LTERM partner is assigned to an LTERM pool.

  ´N´ The LTERM partner is not assigned to an LTERM pool.

out_queue
: The number of asynchronous messages presently in the message queue of the LTERM partner for outputting.

  If the number of messages is greater than 99999, then the number is not displayed in full. You should therefore use the field *out_queue_ex* since larger numbers can be entered in full here.

incounter
: The number of messages entered via this LTERM partner; if a printer is connected via this LTERM partner, then the number of print confirmations from the printer is entered here.
  The *incounter* counter is reset to 0 at every start of the application.

seccounter
: The number of security violations by users and clients that were connected to the application via this LTERM partner (for example, due to entering an unauthorized transaction code).
  The counter is reset to 0 at every start of the application.

deleted
: Specifies whether or not the LTERM partner was dynamically deleted from the configuration.

  ´Y´ The LTERM partner was deleted. No more clients or printers may be connected to the application via this LTERM partner.

  ´N´ The LTERM partner was not deleted.

nbr_dputs
: Number of time-controlled jobs for this LTERM partner whose start time has not yet been reached

lt_group
: Specifies whether the LTERM is a "normal" LTERM, part of an LTERM bundle or part of an LTERM group.

  ´ ´ The LTERM is not part of an LTERM bundle or an LTERM group.

  'P' The LTERM is the primary LTERM of an LTERM group.

  'A' The LTERM is an alias LTERM of an LTERM group.

out_queue_ex
: see *out_queue.*

B kerberos_dialog

B Y When the connection is established, a Kerberos dialog is conducted for
B clients that support Kerberos and are directly connected with the application
B via this LTERM partner (not via OMNIS).
B N No Kerberos dialog is performed.

kc_message_module_str - User message modules

The data structure *kc_message_module_str* is defined for the object type KC_MESSAGE_MODULE. In the case of KC_GET_OBJECT, UTM returns the properties of the user-defined UTM message module of the application in *kc_message_module_str*.

B In UTM applications on BS2000 systems, you can create several user-defined message modules that contain the UTM messages in various languages for the purpose of internationalization of the application. A language code and a territory code are assigned to each UTM message module to precisely define the language. The combination of the language code and the territory code must be assigned to exactly one UTM message module of the application. Through the "language and territory code" the user-defined UTM message modules are assigned to the users and LTERM partners whose locale contains the same language and territory code.

X/W In UTM applications running on Unix, Linux or Windows systems, you can create user-defined message modules. UTM only returns the name of the message module. The other fields of the data structure are of no relevance.

User-defined UTM message modules are defined in the KDCDEF generation with a MESSAGE statement.

How a user-defined UTM message module is created is described in the openUTM manual ”Messages, Debugging and Diagnostics”.

Data structure kc_message_module_str

```
char mm_name[8];
B  char lib[54];
B  char locale_lang_id[2];
B  char locale_terr_id[2];
B  char standard_module;
```

The fields in the data structure have the following meanings:

mm_name
: Contains the name of the UTM message module whose properties are returned by UTM. The name is a maximum of 8 characters long.

B lib
: Contains the name of the library that contains the UTM message module. The name can be up to 54 characters long.

B locale_lang_id, locale_terr_id
: Specifies the language environment for which the UTM message module will be used.

B *locale_lang_id*
: Contains the up to two characters long language code.

B *locale_terr_id*
: Contains an up to two characters long territory code.

B The UTM messages of user-defined UTM message modules are used for the STATION, SYSLINE and PARTNER message lines. The UTM message module used corresponds to the *locale_lang_id* and *locale_terr_id* that is identical to the language and territory code of the locale of the respective user or LTERM partner.

B standard_module
: Specifies if the message module is the user-defined standard message module of the application.

B The standard message module is the user-defined message module that is assigned to the language and territory code of the standard language environment. The standard language environment is specified in the KDCDEF generation in MAX LOCALE.
The standard message module is always used by UTM for messages in the SYSLST, SYSOUT and CONSOLE message lines

B ´Y´ The message module is the standard message module.

B ´N´ The message module is not the standard message module.

B

kc_mux_str - Multiplex connections (BS2000 systems)

B The data structure *kc_mux_str* is defined for the object type KC_MUX. In the case of KC_GET_OBJECT, UTM returns the names and properties of a multiplex connection via which a message router can connect to the application in *kc_mux_str*.

B Several terminal clients can be connected simultaneously to the UTM application via a multiplex connection.

mod[1]	Data structure kc_mux_str
–	`char mx_name[8];`
–	`char pronam[8];`
–	`char bcamappl[8];`
x(GPD)	`char auto_connect;`
x(GPR)	`char maxses[5];`
x(GPD)	`char state;`
–	`char netprio;`
x(A)	`char connect_mode;`
–	`char actcon[5];`
–	`char maxcon[5];`
–	`char letters[10];`
–	`char incnt[5];`
–	`char wait_go[5];`
–	`char shortage[5];`
–	`char rtryo[5];`
–	`char rtryi[5];`
x(IR)	`char bcam_trace;`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 336f

The fields in the data structure have the following meanings:

mx_name
: Contains the name of the multiplex connection.

pronam
: The name of the computer containing the message router.

  The name in *pronam* is identical to the name of the computer specified for the BCAM generation for this computer.

bcamappl
: The name of the local UTM application (BCAMAPPL name) via which the connection to the message router will be established, i.e. the message router must specify this application name as the partner name when the connection to the UTM application is established.

  If several multiplex connections (the same computer name is always in *pronam*) with different BCAMAPPL names exist in the local application for a message router, then several parallel connections can be established to the message router.

auto_connect
: Specifies if the local application automatically establishes a transport connection to the message router during the application start.

  ´N´ The connection is not automatically established, it must be established by the administrator (see *connect_mode*, page 584).

  ´Y´ UTM attempts to establish the connection to the message router at the start of the local application.
  If no connection can be made, for example because the message router is not available, then UTM will repeat the attempt to establish the connection at the intervals specified in the *conrtime_min* timer. The timer can be changed (see the data structure *kc_timer_par_str*, *conrtime_min* field page 656).

maxses
: Specifies the maximum number of simultaneously open sessions that can exist between the message router and the application, i.e. *maxses* contains the maximum number of clients that can be simultaneously connected to the application via the message router.

  Minimum value: ´1´
  Maximum value: ´65000´ (theoretical value)

state Specifies whether the multiplex connection is currently disabled.

´Y´ The multiplex connection is not disabled.

´N´ The multiplex connection is disabled. No connection between the message router and the application can be established at the present time.

netprio
Specifies the transport priority used on the transport connection between the application and the message router.

´M´ "Medium" transport priority

´L´ "Low" transport priority

connect_mode
Specifies whether the message router is currently connected to the application.

´Y´ The message router is currently connected to the application.

´W´ UTM is attempting to establish a connection to the message router.

´N´ The message router is not currently connected to the application.

actcon Contains the number of clients currently connected to the application via this multiplex connection.

maxcon
Contains the maximum value that *actcon* has reached during the current application run. *maxcon* also specifies the maximum number of clients that were simultaneously connected to the application via this multiplex connection during the previous application run.
The counter is reset to 0 at the start of the application.

letters Contains the number of messages replaced between the message router and the application since the start of the application (input and output messages).

incnt Contains the number of input messages received from the application via this multiplex connection.
The counter is reset to 0 at the start of the application.

wait_go
Specifies how often BCAM needed to request the multiplex connection to resend a message because BCAM was not able to accept this message before due to a BCAM bottleneck (WAIT FOR GO).
The counter is reset to 0 at the start of the application.

B shortage
B Contains the number of BCAM bottlenecks (shortages) for this multiplex connection
B since the start of the application.

B rtryo (**re**t**ry o**ut)
B Specifies how often the application needed to retry sending an output message to
B the message router since the application start.

B rtryi (**re**t**ry i**n)
B Specifies how often the application needed to retry reading a message from the
B message router since the application start.
B If a message from the message router is received by BCAM, then BCAM informs
B UTM that a message is available. UTM then tries to read the message from BCAM.
B *rtryi* contains the number of failed attempts to read the message from BCAM before
B UTM was finally able to read the message.

B bcam_trace
B Specifies whether the BCAM trace for this multiplex connection is explicitly
B activated or deactivated.

B ´Y´ The BCAM is explicitly activated.

B ´N´ The BCAM trace is not explicitly activated.

B There is only any point evaluating this field if the BCAM trace is activated explicitly
B for a number of LTERM partners, LPAP partners or multiplex connections.

B If the BCAM trace is activated or deactivated generally
B (*kc_diag_and_account_par_str*), ´N´ is returned for *bcam_trace*.

B If the value of *bcam_trace* is to be modified, the following prerequisites apply to
B explicit activation:

B – The BCAM trace must be deactivated for everything (*kc_diag_and_account_par*).

B – The BCAM trace must be deactivated explicitly for this multiplex connection.

B The prerequisite for explicit deactivation is that the BCAM trace is activated
B explicitly for a number of LTERM partners, LPAP partners or multiplex connections.

kc_osi_association_str - Associations to OSI TP partner applications

The data structure *kc_osi_association_str* is defined for the object type KC_OSI_ASSOCIATION. In the case of KC_GET_OBJECT, UTM returns the properties of an association currently existing or being established for distributed processing via OSI TP in *kc_osi_association_str*.

Data structure kc_osi_association_str

```
char association_id[8];
char osi_lpap[8];
char contwin;
char connect_state;
char contime_min[10];
char request_calls[10];
char indication_calls[10];
```

The fields in the data structure have the following meanings:

association_id
: Contains the identification (ID) assigned to the association when the connection is established. It is only unique as long as the association is established. If this association is closed, then the ID is released and can be assigned to another association (established thereafter).

  The association ID is an integer with a maximum of 8 digits.

osi_lpap
: Specifies the partner application with which the association has been established. UTM returns the name of the OSI-LPAP partner assigned to the partner application in *osi_lpap*.

contwin (**cont**ention **win**ner)
: Specifies if the local application for this association is the contention winner or the contention loser.

  The contention winner takes over the administration of the association. Jobs can be started, however, by the contention winner as well as by the contention loser. In case of a conflict, such as when both communication partners want to start a job at the same time, the association from the job of the contention winner will be used.

  ´Y´ The local application is the contention winner.

  ´N´ The local application is the contention loser.

connect_state
: Specifies the status of the association.

  ´C´ The association is established.

  ´W´ The association is being created. It is waiting for a "GO" from OSS.

  ´S´ The association is being created and is in "STOP" state. It is waiting for a "GO" signal from OSS.

contime_min
: Specifies the connect time of the existing connection in minutes.

request_calls
: The number of request/response presentation calls to OSS since the creation of the association.

indication_calls
: The number of indication/confirmation presentation calls to OSS since the creation of the association.

kc_osi_con_str - OSI TP connections

The data structure *kc_osi_con_str* is defined for the object type KC_OSI_CON. In the case of KC_GET_OBJECT, UTM returns the name and address of an OSI TP partner application and the status of the connection to the partner application in *kc_osi_con_str*.

An OSI TP connection is created with the KDCDEF control statement OSI-CON.

	mod[1]	Data structure kc_osi_con_str
	–	`char oc_name[8];`
	–	`char osi_lpap[8];`
	–	`char local_access_point[8];`
	–	`union kc_selector presentation_selector;`
	–	`union kc_selector session_selector;`
	–	`char presentation_selector_type;`
	–	`char presentation_selector_lth[2];`
	–	`char presentation_selector_code;`
	–	`char session_selector_type;`
	–	`char session_selector_lth[2];`
	–	`char session_selector_code;`
	–	`char transport_selector[8];`
	–	`char network_selector[8];`
	x(GIR)	`char active;`
X/W	–	`char map;`
X/W	–	`char listener_port[5];`
X/W	–	`char t_prot;`
X/W	–	`char tsel_format;`
X/W	–	`char ip_addr[15];`
X/W	–	`char ip_addr_V6[39];`
X/W	–	`char ip_v[2];`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 338

The fields in the data structure have the following meanings:

oc_name
: Contains the name of a connection that was generated with OSI-CON for the communication via the OSI TP protocol. *oc_name* uniquely identifies the connection in the local UTM application. *oc_name* is up to 8 characters long.

osi_lpap
: Specifies the partner application for which the connection is defined. *osi_lpap* contains the name of the OSI-LPAP partner assigned to the partner application. The name can be up to 8 characters long.

local_access_point
: Contains the name of an OSI TP access point that is defined for the local application (KDCDEF statement ACCESS-POINT). The connection to the partner application is established via this access point. The name is a maximum of 8 characters long.

presentation_selector
: Contains the presentation selector of the partner application. The presentation selector is a component of the partner address.

  presentation_selector is a field of type *kc_selector*:

union kc_selector
`char x[32];`
`char c[16];`

  UTM generally returns the presentation selector as character string (*c*) in a machine-specific code format (*presentation_selector_code*=´S´). The character string is a maximum of 16 characters long. The *presentation_selector* field is padded with blanks starting after the position specified in the *presentation_selector_lth* length field.

  In special cases, the presentation selector is returned as a hexadecimal string (*x*). Each half byte is represented by a character, for example the hexadecimal number A2 is returned as the string ´A2´ (2 characters). If the presentation selector is a hexadecimal number, then UTM returns up to 32 bytes.

  You determine how to interpret the contents of the *presentation_selector* with the *presentation_selector_type* field.

  If the address of the access point does not contain a presentation selector, then the *presentation_selector* field contains only blanks. In this case, *presentation_selector_type* = ´N´ and *presentation_selector_lth* = ´0´.

session_selector
: Contains the session selector of the partner application. The session selector is a component of the partner address.

session_selector is a union of type *kc_selector (see presentation_selector*).

UTM generally returns the session selector as character string (*c*) in machine-specific code (*session_selector_code*=´S´). The character string is a maximum of 16 characters long. The *session_selector* field is padded with blanks starting after the position specified in the *session_selector_lth* length field.

In special cases, the session selector is returned as a hexadecimal string (*x*). Each half byte is represented by a character. If the session selector is a hexadecimal number, then UTM returns up to 32 bytes in *session_selector*.

You determine how to interpret the contents of the *session_selector* with the *session_selector_type* field.

If the address of the access point does not contain a presentation selector, then the *session_selector* field contains only blanks. In this case, *session_selector_type* = ´N´ and *session_selector_lth* = ´0´.

presentation_selector_type

Specifies if the address of the partner application contains a presentation selector and how to interpret the data returned in *presentation_selector*.

´N´ N stands for *NONE. The address of the partner application does not contain a presentation selector, *presentation_selector* contains only blanks and *presentation_selector_lth*=´0´.

´C´ The data of the presentation selector in *presentation_selector* is to be interpreted as a character string. A maximum of the first 16 bytes of *presentation_selector* contain data.

´X´ The presentation selector in *presentation_selector* is a hexadecimal number.

presentation_selector_lth

Contains the length of the presentation selector (*presentation_selector)* in bytes. If *presentation_selector_lth* =´0´, then the address of the partner application does not contain any presentation components (*presentation_selector* contains blanks). Otherwise, the value of *presentation_selector_lth* lies between ´1´ and ´16´.

If *presentation_selector_type*=´X´, then the string length specified in *presentation_selector* is: 2 ∗ *presentation_selector_lth* bytes.

Example

The presentation selector is X´A2B019CE´. *presentation_selector* then contains the string ´A2B019CE´, *presentation_selector_type*=´X´ and *presentation_selector_lth* =´ 4´.

presentation_selector_code

Specifies how the presentation selector in *presentation_selector* is encoded.

UTM returns ´S´ if the presentation selector will be returned as a character string (*presentation_selector_type* = ´C´).
´S´ means: machine-specific code (default code, EBCDIC on BS2000 systems and ASCII on Unix, Linux and Windows systems).

If *presentation_selector_type* = ´X´ or ´N´, then UTM returns a blank in the *presentation_selector_code* field.

session_selector_type
: Specifies if the address of the partner application contains a session selector and how to interpret the data returned in *session_selector*.

  ´N´ N stands for *NONE. The address of the partner application does not contain a session selector, *session_selector* contains only blanks and *session_selector_lth*=´0´.

  ´C´ The data of the session selector in *session_selector* is to be interpreted as a character string. A maximum of the first 16 bytes of *session_selector* contain data.

  ´X´ The session selector in *session_selector* is a hexadecimal number.

session_selector_lth
: Contains the length of the session selector (*session_selector)* in bytes. If *session_selector_lth* =´0´, then the address does not contain any session components (*session_selector* contains blanks). Otherwise, the value of *session_selector_lth* lies between ´1´ and ´16´.

session_selector_code
: Specifies how the session selector in *session_selector* is encoded.

  UTM returns ´S´ if the session selector will be returned as a character string (*session_selector_type* = ´C´).
  ´S´ means: machine-specific code (default code, EBCDIC on BS2000 systems and ASCII on Unix, Linux and Windows systems).

  If *session_selector_type* = ´X´ or ´N´, then UTM returns a blank in the *session_selector_code* field.

transport_selector
: Contains the transport selector of the address of the partner application. The transport selector is a component of the partner address. *transport_selector* always contains a valid value because each communication partner must be assigned a transport selector so that it is addressable within its system. The transport selector is always to be interpreted as a character string and consists of 1 to 8 printable characters.

network_selector

Network component (network selector) of the partner address. *network_selector* always contains a valid value.

B *BS2000 systems:*

B *network_selector* contains the name of the computer on which the partner application
B runs. This is the name under which the computer is known to BCAM.

X/W *Unix, Linux or Windows systems:*

X/W *network_selector* contains the official host name under which the IP address of the
X/W partner computer is stored in the name service of the local system (e.g. hosts file).

The network selector consists of a maximum of 8 printable characters.

active Specifies if this connection is set to active or if the connection is a substitute connection that is presently inactive. It is possible to generate several connections to a partner application. Only one of these connections, however, may be active at any one time.

´Y´ The connection is set to active.

´N´ The connection is inactive.

X/W map Specifies whether or not UTM performs ASCII/EBCDIC conversions when
X/W exchanging messages encoded in accordance with the abstract syntax UDT.

X/W ´U´ UTM does not execute any message handling, i.e. the data in the message
X/W is transmitted unchanged to the partner application.

X/W ´S´ UTM converts the data in the messages exchanged between the partner
X/W application and the local application.
X/W The data in a message is converted from ASCII to EBCDIC before sending
X/W to the partner application.
X/W Data in the messages received is converted from EBCDIC to ASCII.
X/W The messages may only contain printable characters.

X/W UTM returns the components of the transport address of the partner application in the
X/W following fields. See also the openUTM manual “Generating Applications” for more infor-
X/W mation.

X/W listener_port
X/W Contains the port number of the address of the partner application.

X/W Possible values of *listener_port*: ´0´, ´102´, ´1025´ to ´32767´

X/W If *listener_port* = ´0´, then no listener port number was generated in the KDCDEF
X/W generation.

X/W t_prot Contains the address format of the transport address. The address format specifies
X/W the transport protocol used for communication with the partner application.

X/W ´R´ RFC1006, ISO transport protocol class 0 using TCP/IP and the conver-
X/W gence protocol RFC1006.

X/W If *t_prot* contains a blank, then no address format was defined in the KDCDEF
X/W generation.

X/W tsel_format
X/W Specifies the format in which the T-selectors of the partner address is stored in the
X/W TS directory:

X/W ´T´ TRANSDATA format
X/W ´E´ EBCDIC character format
X/W ´A´ ASCII character format

X/W If *tsel_format* contains a blank, then no format indicator was defined in the KDCDEF
X/W generation.

X/W The meanings of the address formats are described in the “PCMX documentation” on page 20.

X/W ip_addr
X/W Returns the IP address used by UTM for this connection from the object table of the
X/W application if the address is an IPv4 address.

X/W UTM uses the address to establish connections to partner applications. UTM reads
X/W the IP address from the host name database when the application is started using
X/W the generated processor name (*networ_selector*).

X/W An IPv6 address is returned in the *ip_addr_v6* field.

X/W If there is no IPv4 address in the object tables for the client, UTM returns blanks in
X/W *ip_addr*.

X/W ip_addr_v6
X/W Returns the IP address used by UTM for this connection from the object table of the
X/W application if the address is an IPv6 address or an IPv4 address embedded in IPv6
X/W format.

X/W An IPv4 address is returned in the *ip_addr* field (see above).

X/W If there is no IPv6 address in the object tables for the client, UTM returns blanks in
X/W *ip_addr_v6*.

X/W ip_v Specifies whether the IP address used by openUTM for this connection is an IPv4
X/W or an IPv6 address:

X/W 'V4' IPv4 Address.
X/W 'V6' IPv6 address or IPv4 address embedded in IPv6 format.

kc_osi_lpap_str - Properties of OSI TP partner applications

The data structure *kc_osi_lpap_str* is defined for the object type KC_OSI_LPAP. In the case of KC_GET_OBJECT, UTM returns the following in *kc_osi_lpap_str*:

– The logical properties of an OSI TP partner application.
  The logical properties of an OSI TP partner application are defined in the KDCDEF generation in which an OSI-LPAP partner is created and assigned to this partner application.
– The status of the connection to the partner application.
– Statistical information on the connection load.

mod[1]	Data structure kc_osi_lpap_str
–	`char ol_name[8];`
–	`char application_context[8];`
–	`char application_entity_qualifier[8];`
–	`char application_process_title[10][8];`
–	`char association_names[8];`
–	`char associations[5];`
x(GPD)	`char auto_connect_number[5];`
–	`char contwin[5];`
–	`char kset[8];`
x(GPD)	`char idletime_sec[5];`
x(GPD)	`char state;`
–	`char permit;`
–	`char qlev[5];`
–	`char termn[2];`
x(A)	`char connect_number[5];`
x(IR)	`char quiet_connect;`
–	`char osi_con[8];`
–	`char out_queue[5];`
–	`char ass_kset[8];`
–	`char nbr_dputs[10];`
–	`char master[8];`
–	`char bundle;`
–	`char out_queue_ex[10];`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 339

The fields in the data structure have the following meanings:

ol_name
: Contains the name of the OSI-LPAP partner of the partner application. The partner application will be addressed by the program units of the local application using this name. The name consists of a maximum of 8 alphanumeric characters.

application_context
: Specifies which application context from the partner application will be used. For details on application context, please refer to the openUTM manual “Generating Applications”, KDCDEF statement APLLICATION-CONTEXT.

application_entity_qualifier
: Contains the application entity qualifier of the partner application. The application entity qualifier is used together with the application process title for addressing a partner application for a heterogeneous link. The application entity qualifier is a positive integer between 1 and 67108863 (= 2^{26}-1).

  You will find more information on the application entity qualifier in the openUTM manual “Generating Applications”.

  application_entity_qualifier=´0´ means that no AEQ is defined for the partner application.

application_process_title
: Contains the application process title (APT) of the partner application. The APT is used together with the application entity qualifier for addressing a partner application for a heterogeneous link.

  An APT consists of at least two, but at most ten components. The individual components are positive integers.

  The position of the individual components as well as their number is relevant in an application process title. For example, (1,2,3), (1,2,3,0,0) and (0,1,2,3,0) indicate different application process titles.

  You will find more information on the application process title in the openUTM manual “Generating Applications”.

  UTM returns one field element per component of the APT, i.e. the number of field elements containing data in *application_process_title* corresponds to the number of components generated. The remaining field elements contain binary zero.

  If no APT was generated for the partner application, then all field elements of *application_process_title* contain binary zero.

association_names
: Contains the prefix of the names that are assigned to the logical connections (associations) to the partner application within the local application.

The name of the connection is composed of the value of *association_names* as its prefix and a sequential number. The sequential number lies between 1 and the number of parallel connections (*associations* field) generated. The entire name for a connection can be up to 8 characters long. The maximum length of *association_names* depends, therefore, on the number of parallel connections in *associations*.

Example
association_names =´ASSOC´ and *associations* = ´10´, then the connections to the partner application that are assigned to the OSI-LPAP partner have the following names: ASSOC01, ASSOC02, ..., ASSOC10.

associations
Contains the maximum number of parallel connections to the partner application. The maximum possible number of parallel connections to a partner application depends on the transport system used and on the size of the name space of the UTM application (see the openUTM manual “Generating Applications”).

auto_connect_number
Contains the number of connections to the partner application that will be automatically established at start of the local application as long as the partner application is available at this point in time. The establishing of a connection at the start of the application can be requested by the local application as well as by the partner application. In this manner, you can ensure that the connection is automatically established as soon as both partners are available.

auto_connect_number = ´0´ means that the connection is not set up automatically.

Minimum value: ´0´
Maximum value: maximum number of parallel connections (*associations*)

contwin
Contains the number of connections for which the local application is the contention winner. For the rest of the connections (difference: *associations* - *contwin*), the local application is the contention loser.

The contention winner takes over the administration of the association. Jobs can be started, however, by the contention winner as well as by the contention loser. In case of a conflict, such as when both communication partners want to start a job at the same time, the association from the job of the contention winner will be used.

Minimum value: 0
Maximum value: maximum number of parallel connections (*associations*)

kset Contains the name of the key set with the maximum access privileges of the OSI TP partner application in the local application.

If the OSI TP partner passed a user ID when an association is established, the key set in *kset* becomes effective. The access privileges of the association correspond to the key codes contained both in *kset* and in the key set of the user ID.

If the OSI TP partner application for the association does not pass a user ID to openUTM the access privileges for the association form the subset of the key codes in *kset* and *ass_kset* (minimum access privileges).

If the partner application is not assigned a key set, then *kset* contains blanks.

idletime_sec
Contains the maximum time in seconds that an association to the partner application may be in the idle state before UTM closes the connection to the partner application. The idle state means that the session is not handling any jobs. *idletime_sec* = ´0´ means that the idle state will not be monitored. The connection remains established until an explicit request to close the connection is sent.

Minimum value: ´60´
Maximum value: ´32767´

state Specifies if the OSI-LPAP partner is currently disabled.

´Y´ The OSI-LPAP partner is not disabled. Connections between the partner application and the local application can be established or connections already exist.

´N´ The OSI-LPAP partner is disabled. No connections between the partner application and the local application can be established.

permit Specifies which privileges the partner application has within the local application.

´A´ (ADMIN)
The partner application has administration privileges. It may execute all administration functions in the local application.

´N´ (NONE)
The partner application does not have any administration privileges.

B B B If the local application is a UTM application on a BS2000 system, then the partner application is also not allowed to execute any UTM SAT administration functions.

B B B ´B´ (BOTH)
The partner application may execute administration functions and UTM SAT administration functions in the local application.

B B B B ´S´ (SAT)
The partner application only has UTM SAT administration privileges. It may execute preselection functions in the local application, i.e. it can enable or disable the SAT logging for certain events.

qlev (**q**ueue **lev**el)
qlev specifies the maximum number of asynchronous messages allowed in the message queue of the OSI-LPAP partner. If this control value is exceeded, then any additional asynchronous jobs sent to this OSI-LPAP partner will be rejected (i.e. ´40Z´ will be returned for any APRO-AM calls thereafter).

termn Contains the code for the type of communication partner. The code is entered in the communication area header of the job-receiving service that was started in the local application by the partner application. The code is defined by the user and serves to divide the communication partners into groups of a certain type. It is not evaluated by UTM. The code is a maximum of 2 characters long.

connect_number
Contains the number of parallel connections to the partner application that are currently established or that are currently to be established.

connect_number = ´0´ means that no connection to the partner application currently exists or all existing connections are to be disconnected.

Minimum value: ´0´
Maximum value: the number in *associations*; a numeric value larger than ´32767´ will be rejected.

quiet_connect
Specifies if the QUIET property is set or is to be set for the partner application. QUIET means that UTM closes all connections to the partner application as soon as they are not being used for dialog jobs or asynchronous jobs. No more new dialog jobs are accepted for the partner application.

´Y´ The QUIET property is set for the partner application.

´N´ The QUIET property is not set.

osi_con
Contains the name of the transport connection via which communication with the partner application will occur, i.e. all connections (associations) with the partner application are handled via this transport connection. The name is assigned to the transport connection in the KDCDEF generation (OSI-CON statement assigned to the OSI-LPAP partner). *osi_con* indicates the transport connection that is currently set to active, i.e. that is not deactivated as a substitute connection (see the openUTM manual “Generating Applications”).

out_queue
The number of messages in the message queue of the OSI-LPAP partners that still have to be sent to the partner application.

If this number of messages is greater than 99999, then the number is not displayed in full. You should therefore use the field *out_queue_ex* since larger numbers can be entered in full here.

ass_kset
: Only applies if the application is generated with user IDs.
ass_kset contains the name of the key set specifying the minimum access privileges of the OSI TP partner which the partner application can use in the local application.

  The key set specifies in *ass_kset* becomes effective when the partner application does not pass a user ID to openUTM while an association is established (see also *kset*). *ass_kset* describes the access privileges of the association user.

  Default: no key set, i.e. the access privileges in *kset* always apply.

nbr_dputs
: The number of pending time-driven jobs for this OSI-LPAP whose starting time has not yet been reached.

master
: Associated OSI-LPAP if the OSI-LPAP is a slave OSI-LPAP in an OSI-LPAP bundle.

bundle
: Specifies whether the OSI-LPAP belongs to an OSI-LPAP bundle.

  ’N’ The OSI-LPAP does not belong to an OSI-LPAP bundle.

  ’M’ The OSI-LPAP is the master OSI-LPAP in an OSI-LPAP bundle. In this event, the following applies:
  – Only the *state* field can be modified with KC_MODIFY_OBJECT.
  – Only the *ol_name*, *application_context*, *state* and *bundle* fields are relevant with KC_GET_OBJECT.

  ’S’ The OSI-LPAP is the slave OSI-LPAP in an OSI-LPAP bundle.

out_queue_ex
: See *out_queue* on page 482.

kc_program_str - Program units and VORGANG exits

The data structure *kc_program_str* is defined for the object type KC_PROGRAM. In the case of KC_GET_OBJECT, UTM returns information in *kc_program_str* on the program units and VORGANG exits of the UTM application.

Program units can be dynamically created with KC_CREATE_OBJECT and deleted with KC_DELETE_OBJECT.

Data structure kc_program_str

```
char pr_name[32];
char compiler;
char load_module[32];
char load_mode;
char poolname[50];
char lib[54];
char deleted;
```

The fields in the data structure have the following meanings:

pr_name

Contains the name of the program unit. The name can consist of up to 32 alphanumeric characters.

B B In UTM applications on BS2000 systems, UTM returns the ENTRY or CSECT name of the program unit.

compiler

Specifies the compiler used to compile the program unit. The values returned by UTM depend on the operating system platform on which the program unit is running.

B B B B B In a UTM applications on BS2000 systems *compiler*=´I´ will be returned for all program units that support ILCS. For program units that do not support ILCS, UTM returns the compiler with which the program unit was compiled. If the C compiler was specified when the program unit was added to the configuration, then UTM always returns *compiler* = ´I´ (ILCS) as UTM maps C to ILCS.

B The following values are possible in a UTM application on BS2000 systems:

B ´I´ for ILCS (Inter Language Communication Services)

B ´A´ for the ASSEMB assembly compiler

B ´C´ for the C compiler (only for KC_CREATE_OBJECT call)

B ´1´ for the COBOL compiler COB1

B ´F´ for the FORTRAN compiler FOR1

B ´X´ for PASCAL-XT

B ´P´ for PLI1

B ´S for SPL4

X/W The following values are possible for a UTM application on Unix, Linux or Windows
X/W systems:

X/W ´C´ for the C compiler

X/W ´2´ for the COBOL compiler of Micro Focus

X/W ´3´ for the NetCOBOL compiler of Fujitsu

X/W ´+´ for the C++ compiler

load_module

Contains the name of the load module (BS2000 systems) or shared object/DLL (Unix, Linux and Windows systems) in which the program unit is bound. *load_module* can be up to 32 characters long.
If the application was generated without load modules or shared objects/DLL, then UTM returns blanks.

load_mode

Contains the load mode of the program unit or of the load module/shared object/DLL in which the program unit is bound. The load mode specifies when and to where the program unit or load module/shared object/DLL will be loaded.

´U´ (STARTUP)
The program unit or load module/shared object/DLL will be loaded as an independent unit at the start of the application.

´O´ (ONCALL)
The load module/shared object/DLL is loaded as an independent unit when one of its VORGANG exits is called for the first time.

B B The following values can additionally be returned in *load_mode* for load modules of a UTM application on a BS2000 system:

B B B ´S´ (STATIC)
The program unit or load module is statically bound in the application program.

B B B B ´P´ (POOL)
The program unit or load module is loaded into a common memory pool (see *poolname*) at the start of the application. The load module consists only of one public slice (no private slice).

B B B B B ´T´ (POOL/ STARTUP)
The public slice of the load module is loaded into a common memory pool (see *poolname*) at the start of the application. The private slice belonging to the load module is then loaded into the local process storage area (private slice with load mode STARTUP).

B B B B B B ´C´ (POOL/ONCALL)
The public slice of the load module is loaded into a common memory pool (see *poolname*) at the start of the application. The private slice belonging to the load module is then loaded into the local process storage area as soon as a program unit is called that is assigned to this load module (private slice with load mode ONCALL).

B B B B poolname
For *load_mode*=´P´, ´T´ or ´C´, UTM returns in *poolname* the name of the common memory pool in which the program unit or the public slice of its load module was loaded at the start of the application. The name is a maximum of 50 characters long.

B For *load_mode* ≠ ´P´, ´T´ or ´C´, *poolname* contains blanks.

lib UTM returns the following in *lib*:

B B B – In a UTM application on a BS2000 system generated without load modules, the object module library from which the program unit was loaded or bound is returned.

B B – In a UTM application on a BS2000 system generated with load modules, the program library from which the load module was loaded is returned.

X/W X/W X/W – In a UTM application running on Unix, Linux or Windows systems generated with shared objects, the directory in which the shared object/DLL is stored is returned.

deleted

Specifies if the program unit was deleted from the configuration by the dynamic administration.

´Y´ The program unit was deleted. The name is disabled, meaning no new program unit with this name may be added.

´N´ The program unit was not deleted from the configuration.

kc_ptc_str - Transactions in PTC state

The data structure *kc_ptc_str* is defined for the object type KC_PTC. If KC_GET_OBJECT is specified then *kc_ptc_str* shows all the distributed transactions in the state PTC (prepare to commit) and for which there is no connection (LU6.1 session or OSI-TP) to the Commit Coordinator. The Commit Coordinator is the partner application that determines the result of the transaction.

openUTM returns the following:

– Information about the transaction
– Job-submitter user ID of the distributed transaction.
– Name of the partner (LPAP or OSI-LPAP partner)
– Name of the session (in the case of LU6.1)

The PTC state can be caused by the establishment of a connection to the specified partner or by an administrative rollback of the local element of the distributed transaction (e.g. with operation code KC_PTC_TA and *subopcode1*=KC_ROLLBACK, see page 401).
Caution: An administrative rollback can lead to data inconsistencies and should only be performed in exceptional cases.

Data structure kc_ptc_str

```
struct kc_ptc_id_str ptc_ident;
char ptc_user[8];
char ptc_lpap[8];
char ptc_lses[8];
char ptc_user_type;
```

The fields in the data structure have the following meanings:

ptc_ident

Specifies information relating to the transaction in the element *ptc_ident* of type *kc_ptc_id_str*:

struct kc_ptc_id_str

```
char vg_indx[10];
char vg_nr[10];
char ta_nr_in_vg[5];
```

vg_indx is the index of the service, *vg_nr* the number of the service and *ta_nr_in_vg* the number of the transaction in the service.

In the case of operation code KC_PTC_TA with *subopcode1*=KC_ROLLBACK the structure must be passed in the identification area if you wish to reset the transaction.

ptc_user
: Specifies the job submitter user ID of the distributed transaction.

  In the case of OSI TP, the field contains the name of an OSI TP association.

  In the case of LU6.1, the field can contain the name of a user (USER), an LU6.1 session (LSES) or 8 spaces:

  – If the field contains 8 spaces then the transaction is in the PTC state in an asynchronous LU6.1 job-submitter service.
  – If the field contains the name of a user then the transaction is in the PTC state in the highest level LU6.1 job-submitter dialog service.
  – If the content of the field is not the same as the content of the *ptc_lses* field then the transaction is in the PTC state in an LU6.1 job-submitter service.
  – If the content of the field is the same as the content of the *ptc_lses* field then the transaction is in the PTC state in an LU6.1 job-receiver service.

ptc_lpap
: Specifies the LPAP or OSI-LPAP name of the partner that determines the result of the transaction (Commit Coordinator).

ptc_lses
: In the case of LU6.1 connections, specifies the session name of the partner that determines the outcome of the transaction (Commit Coordinator).

  In the case of a PTC transaction in the job receiver, *ptc_lses* has the same content as *ptc_user*.

  In the case of OSI TP connections, the field contains spaces.

ptc_user_type
: Type of entry in the field *ptc_user*:

U	User
L	LU6.1 session
O	OSI TP Association
Blank	If the *ptc_user* field is empty

kc_pterm_str - Clients and printers

The data structure *kc_pterm_str* is defined for the object type KC_PTERM. In the case of KC_GET_OBJECT, UTM returns the following information in *kc_pterm_str*:

- The properties of clients and printers that were statically or dynamically added to the configuration of the application.
- The properties of clients that are presently connected to the application via an LTERM pool or multiplex connection.
- The properties and status of the connection to the corresponding client or printer.
- Statistical information on the connection load and the demands on the application for the individual clients/printers.

Clients and printers can be dynamically created with KC_CREATE_OBJECT, deleted with KC_DELETE_OBJECT or modified with KC_MODIFY_OBJECT.

	mod[1]	Data structure kc_pterm_str	Page[2]
	–	`char pt_name[8];`	530
	–	`char pronam[8];`	530
	–	`char bcamappl[8];`	531
	–	`char ptype[8];`	531
	–	`char ptype_fotyp[8];`	534
	–	`char ptype_class[40];` [3]	
	x(PD)	`char lterm[8];`	534
	x(GPD)	`char auto_connect;`	534
	x(GPD)	`char state;`	534
	–	`char cid[8];`	535
	–	`char map;`	535
	–	`char termn[2];`	536
B	–	`char protocol;`	536
B	–	`char usage_type;`	536
	–	`char listener_port[5];`	537
	–	`char t_prot;`	537
X/W	–	`char tsel_format;`	537
	x(A)	`char connect_mode;`	537
	–	`char pool;`	538
B	–	`char mux;`	538

mod[1]	Data structure kc_pterm_str	Page[2]
–	`char contime_min[10];`	538
–	`char letters[10];`	538
–	`char conbad[5];`	538
–	`char deleted;`	538
X(GPD)	`char idletime[5];`	539
–	`char encryption_level;`	538
–	`char ip_addr[15];`	538
–	`char curr_encryption;`	538
–	`char t_mode;` [3]	
–	`char usp_hdr;`	541
–	`char ip_addr_v6[39];`	542
–	`char ip_v[2];`	542

[1] Field contents can be modified with KC_MODIFY_OBJECT; see page 343.

[2] The meaning of the fields is described on the pages indicated in this column.

[3] UTM-internal field; the field contents is irrelevant and will not be discussed.

The fields in the data structure have the following meanings:

pt_name
: Contains the name of the client or printer. The client/printer is known to the transport system (BCAM, PCMX) under this (symbolic) name. The name is a maximum of 8 characters long.

pronam
: The name of the computer on which the client can be found or to which the printer is connected.

B B B B
: In UTM applications on BS2000 systems *pronam* always contains data. The name of the computer in *pronam* is identical to the name of the computer specified for the BCAM generation or in the RTF file for this computer. For an RSO printer *pronam* contains the value ´*RSO´.

X/W X/W
: In UTM applications running on Unix, Linux or Windows systems *pronam* contains blanks for a local client or printer.

bcamappl

Name of the local UTM application via which the connection to the client/printer will be established.

For terminals and printers *bcamappl* always contains the name of the application that was specified for the KDCDEF generation in MAX APPLINAME.

For UPIC clients and TS applications *bcamappl* always (even if no connection is presently established) contains the application name assigned to the client when it was added to the configuration.

B B B For clients that are connected to the application via a multiplex connection, *bcamappl* contains the application name assigned to the multiplex connection when it was added to the configuration as long as the connection is established.

ptype The type of client or printer.

B B B For clients/printers that are connected to a UTM application on a BS2000 system, either the partner type or the value ´*ANY´ or ´*RSO´ is returned. The partner types supported are listed in the following table:

ptype	Type of client/printer	termn field
*ANY	The client was added to the configuration without an exact specification of its device type. In this case, UTM uses the device type of the client from the user service log when establishing the connection. Only then will it be decided if the partner type is supported or not. Advantage of *ptype*=*ANY: You can add clients to the configuration without knowing their type. In addition, the administration of the configuration is made easier because even if the type is modifieid in the configuration, for example, this client will still be able to establish a connection to the application without you having to change the configuration of the application.	If the terminal mnemonic was not explicitly specified during configuration, then the standard terminal mnemonic of the partner type is used when establishing the connection. Otherwise, the value specified during configuration is stored here.
T100	Teletype T100	C0
T1000	Teletype T1000	E1
T8103	8103	FD
T8110	8110	C2
T8121V12	Printer 8121 on 8112	F7
T8122V12	Printer 8122 on 8112	F8
T8124	Printer 8124	FC
T8151	DSS 8151	F1
T8152	DSS 8152	F2

	ptype	Type of client/printer	termn field
B	T8160	DSS 8160	F4
B	T8162	DSS 8162	F6
B	T8167	DSS 8167	FB
B	T9748 [1]	DSS 9748	FE
B	T9749	DSS 9749	FE
B	T9750 [1]	DSS 9750	FE
B	T9751	DSS 9751	FE
B	T9752	DSS 9752	FF
B	T9753	DSS 9753	FE
B	T9754	DSS 9754	FI
B	T9755 [2]	DSS 9755	FG
B	T9756 [2]	DSS 9756	FG
B	T9763	DSS 9763	FH
B	T9770	DSS 9770	FK
B	T9770R	DSS 9770R	FK
B	T3270	DSS 3270 (IBM)	FL
B	THCTX28	DSS X28 (TELETYPE)	C5
B	TVDTX28	DSS X28 (VIDEO)	C6
B	TPT80	Data station PT80	C4
B	T9001	9001 printer	C7
B	T9002	9002 printer	FA
B	T9003	9003 printer	F9
B	T9004	9004 printer	FD
B	T9001-3	9001-3 printer	CA
B	T9001-89	9001-893 printer	CB
B	T9011-18	9011-18 printer	CC
B	T9011-19	9011-19 printer	CD
B	T9012	9012 printer	CE
B	T9013	9013 printer	C9
B	T9021	9021 printer	CH
B	T9022	9022 printer	CF
B	T3287	3287 printer	CG

ptype	Type of client/printer	termn field
*RSO	Printer supported by RSO. Instead of establishing a transport connection, UTM reserves the printer with RSO and transmits the message to be printed to RSO.	RS
THOST	Intelligent terminal	A3
APPLI	Transport system application that is not a socket application (e.g. : DCAM, CMX or UTM application)	A1
SOCKET	Socket application	A7
UPIC-R	UPIC client	A5

[1] T9748 and T9750 designate the same terminal type.

[2] T9755 and T9756 designate the same terminal type.

The VTSU versions that support the individual terminals can be obtained from the DCAM, FHS or TIAM Manual.
If a terminal is not supported by VTSU, then UTM rejects a request for connection from this terminal. UTM triggers the UTM messages K064 and K107.

X/W
X/W

In a UTM application on Unix, Linux or Windows systems, *ptype* can contain the following values.

X/W

ptype	Type of client/printer	termn field
TTY	The client is a terminal. Default value	F1
PRINTER	The significance of *ptype* =´PRINTER´ depends on the contents of the *ptype_fotyp* field. *ptype_fotyp* contains blanks: It is a printer for a spool on Unix or Linux systems (*lpr*). *pt_name* specifies a spool queue or a printer group to which the spool on Unix or Linux systems (*lpr*) will print. *ptype* =´PRINTER´ is not allowed under openUTM on Windows systems.	F2
APPLI	The client is a TS application that does not use the socket interface (e.g. UTM, CMX, or DCAM application).	A1
SOCKET	The client is a socket application.	A7
UPIC-L	The client is a local Client application with the UPIC carrier system.	A5
UPIC-R	The client is a remote Client application with the UPIC carrier system.	A5

lterm Name of the LTERM partner assigned to this client/printer. The name is a maximum of 8 characters long.

auto_connect
Specifies if the connection to the client/printer will be automatically established at the start of the application:

´Y´ When starting the application, UTM attempts to establish the connection automatically if the client/printer is available when the local application is started.

´N´ No automatic establishing of the connection when starting.

state Specifies if the client or printer is currently disabled.

´Y´ The client/printer is not disabled, i.e. as long as the LTERM partner assigned to this client/printer is not disabled, connections between the client/printer and the local application can be established, or there are already established connections.

´N´ The client/printer is disabled. No connections between the client/printer and the local application can be established.

cid (**c**ontrol **id**entification)
Only contains data if information about a printer is requested. *cid* contains the printer ID (CID) as long as a CID was assigned to the printer when it was added to the configuration. The CID is a maximum of 8 characters long.

The CID has the following function:
– Using the CID, the printer can be identified at the program interface for the purpose of printer control.
– If the printer is assigned to a printer control LTERM, then the printer will be identified by the administration from the printer control LTERM using the CID.

map Specifies whether or not UTM converts user messages before transferring them to or receiving them from a partner.

´U´ (USER)
UTM does not convert any data, i.e. the data in the message is transmitted unchanged to the partner application.

B ´S´, ´1´, ´2´, ´3´, ´4´
B Only possible if communication to the TS application is realized via the
B socket interface (SOCKET).

B The data pertaining to a message is converted from EBCDIC to ASCII
B before being sent and from ASCII to EBCDIC after being received. In this
B context, UTM assumes that the message contains printable characters only.

B With ´S´ and ´1´, data is converted using the standard code tables, i.e. from
B EBCDIC to 7-bit ASCII. 2´, ´3´ and ´4´ stand for the three conversion tables
B that you can define yourself (see the openUTM manual “Generating Appli-
B cations”, keyword „code conversion tables“).

X/W ´S´ is allowed on Unix, Linux and Windows systems only if the client is a TS
X/W application (*ptype*=´SOCKET´ or ´APPLI´).

X/W With ´S´, UTM converts the data in a message being passed between a
X/W partner application and a local application. Before a message is sent, the
X/W data is converted from ASCII to EBCDIC, and after it has been received it
X/W is converted from EBCDIC to ASCII. In this context, UTM assumes that the
X/W message contains printable characters only.

X/W On Unix, Linux and Windows systems, UTM never performs message
X/W handling for formatted messages.

termn (**ter**minal **mn**emonic)

Contains the code for the type of communication partner. UTM KDCS program units provide the code in the KCTERMN field of the communication area header. The code is a maximum of 2 characters long.
The values that *termn* may contain can be obtained from the table for *ptype* on page 531 (BS2000 systems) or on page 534 (Unix, Linux and Windows systems).

B protocol

Specifies if the NEABT user protocol service will be used on connections between the UTM application and the client/printer.

B ´N´ The user protocol service will not be used between the UTM application and the client/printer.

protocol=´N´ will always be output for:
- UPIC clients (*ptype*=UPIC-R)
- TS applications (*ptype*=APPLI or SOCKET)
- printers access via RSO (*ptype*=*RSO)

Clients for which *protocol*=´N´ is specified cannot connect to the application over a multiplex connection.

B ´S´ (STATION)
The user protocol service (NEABT) is used between the UTM application and the client/printer.

UTM uses the user protocol service NEABT, for example, to determine the type (*ptype*) of a client or printer if the type was not explicitly specified when the client/printer was added to the configuration (added with *ptype*=*ANY). In this case, NEABT is always used.

B usage_type

Specifies if the communication partner in *pt_name* is a dialog partner or purely an output medium.

B ´D´ The client is a dialog partner. The client as well as the local application can send messages on the connections between the client and the local application.
UPIC clients are always dialog partners (*ptype*=UPIC-R).

B ´O´ The client/printer is used purely as an output medium. Messages can only be sent from the application to the client/printer. *usage_type*=´O´ is always output for printers.

listener_port

B Only relevant if *t_prot*='T'.

X/W Only relevant if *t_prot*='T' or 'R'.

listener_port contains the port number of the partner application in the remote system in TCP/IP connections. The port number defined when the client was generated is returned.

X/W Possible values of *listener_port*: ´0´, ´102´, ´1025´ to ´32767´

If *listener_port* = ´0´, then no listener port number was generated.

t_prot Contains the address format that the client uses to sign on to the transport system.

The following address formats are possible:

´T´ native TCP/IP transport protocol TCP-IP for communication via the socket interface (SOCKET)

X/W ´R´ RFC1006, ISO transport protocol class 0 using TCP/IP and the convergence protocol RFC1006.

X/W If the client was not assigned an address format when added to the configuration, then the *t_prot* field contains a blank.

X/W tsel_format

X/W Contains the format indicator of the T-selector in the address of the client.

X/W The following format indicators may occur:

X/W ´T´ TRANSDATA format

X/W ´E´ EBCDIC character format

X/W ´A´ ASCII character format

X/W The meanings of the address formats are described in the “PCMX documentation” on page 20.

X/W If the client was not assigned a format indicator when added to the configuration, then the *tsel_format* field contains a blank.

connect_mode

Specifies the current status of the connection to the client:

´Y´ The connection is established.

´W´ UTM is currently attempting to establish the connection (waiting for connection).

´N´ The connection is not established.

B In UTM applications on BS2000 systems *connect_mode* can also contain the following values for clients/printers that are connected to the application via a multiplex connection (*mux* = ´Y´):

B ´T´ (timer)
The session with the client is in the DISCONNECT-PENDING state; the timer used for timing the wait for the confirmation of the closing of the connection is running.

B ´E´ (expired)
The session is in the DISCONNECT-PENDING state and the timer used for timing the wait for the confirmation of the closing of the connection has run down before the confirmation was received. The session must be explicitly released (for example using KC_MODIFY_OBJECT and *connect_mode* = ´R´,see page 345).

pool Specifies if the client is connected to the application via an LTERM pool.

´Y´ The client is connected to the application via an LTERM pool.

´N´ The client is not connected to the application via an LTERM pool.

B mux Specifies if the client is connected to the application via a multiplex connection.

B ´Y´ The client is connected to the application via a multiplex connection.

B ´N´ The client is not connected to the application via a multiplex connection.

contime_min
Specifies how long the connection to the client/printer has already existed. The length of time is specified in minutes.

letters Contains the number of input and output messages for the client/printer since the last start of the local application.

conbad
Specifies how often the connection to the client/printer has been lost since the last start of the application.

deleted
Specifies whether or not the client/printer has been deleted from the configuration.

´Y´ The client/printer has been deleted. The name, however, is disabled, i.e. no new client/printer can be added using this name.

´N´ The client/printer has not been deleted.

idletime

Only relevant for dialog partners.

idletime contains the time in seconds, which UTM waits for a response from a client, after a transaction is terminated or after sign-off. If the time is exceeded the connection to the client is closed down. If the client is a terminal, the message K021 was issued before connection close-down.
0 means indefinite wait.

encryption_level

Only relevant for UPIC clients and, on BS2000 systems, with some terminal emulations.
encryption_level specifies whether the UTM application demands by default for all messages on the connection to the client

– to be encrypted or not,
– which encryption level is demanded,
– or whether the client is a “trusted” client.

The following values are possible:

´N´ (NONE)
UTM does **not** demand the messages exchanged between the client and the UTM application to be encrypted by default.
Services for which encryption was generated (see *kc_tac_str.encryption_level* on page 556) can only be started by a client if the client explicitly selects encryption.

´1´ (LEVEL 1)
UTM demands by default the encryption of messages with encryption level 1. In other words, the messages are encrypted with the DES algorithm and an RSA key with a key length of 200 bits is used for exchange of the DES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´2´ (LEVEL 2)
UTM demands by default the encryption of messages with encryption level 2. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 512 bits is used for exchange of the AES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´3´ (LEVEL 3)
UTM demands by default the encryption of messages with encryption level 3. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 1024 bits is used for exchange of the AES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´4´ (LEVEL 4)
UTM demands by default the encryption of messages with encryption level 4. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 2048 bits is used for exchange of the AES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´T´ (TRUSTED)
The client is a “trusted” client.
Messages exchanged between the client and the application are not encrypted.
A “trusted” client can also start services for which the service TAC requires encryption (generated with *kc_tac_str.encryption_level* =´1´ or ´2´; see page 556).

ip_addr

Returns the partner IP address used by UTM for this connection if the address is an IPv4 address.

B
B On BS2000 systems, IP adresses are output only for partners where *ptype*='SOCKET' ,

An IPv6 address is returned in the *ip_addr_v6* field (see page 542)

In *ip_addr*, UTM returns the IP address of the client computer. The address is stored in the object table of the application. UTM uses this address to establish the connection to the client. UTM reads the IP address from the name service with the aid of the generated processor name (*pronam*) when the application is started.

If the object table does not contain an IPv4 address for the client, e.g. because the client does not use the appropriate protocol, UTM returns blanks in *ip_addr*.

curr_encryption

Only relevant for UPIC clients and on BS2000 systems for some terminal emulations.
In *curr_encryption*, UTM returns the encryption level for an existing connection to a client which was agreed between the UTM application and the client for this specific connection. For information on the properties of encryption levels 1 to 4 see also page 539.

´N´ (NONE)
Messages exchanged on this connection are not encrypted.

´1´ (LEVEL 1)
All messages on the connection are encrypted. Encryption level 1 is used.

´2´ (LEVEL 2)
All messages on the connection are encrypted. Encryption level 2 is used.

´3´ (LEVEL 3)
All messages on the connection are encrypted. Encryption level 3 is used.

´4´ (LEVEL 4)
All messages on the connection are encrypted. Encryption level 4 is used.

´ ´ (Blank)
There is currently no connection to this client.

usp_hdr

This is only relevant for socket partners.
It indicates for which output messages UTM sets up a UTM socket protocol header on this connection. Possible values are:

´A´ UTM creates a UTM socket protocol header for all output messages (dialog, asynchronous, K messages) and precedes the message with it.

´M´ UTM creates a UTM socket protocol header for the output of K messages and precedes the message with it.

´N´ UTM does not create a UTM socket protocol header for any output message.

ip_addr_v6

Returns the partner IP address used by UTM for this connection if the address is an IPv6 address or an IPv4 address embedded in an IPv6 format.

B
B

On BS2000 systems, IP adresses are output only for partners where *ptype*='SOCKET' ,

An IPv6 address is returned in the *ip_addr_v6* field (see page 542)

UTM returns the IP address of the client computer stored in the object table of the application in *ip_addr_v6*. UTM uses this address to establish the connection to the client. UTM reads the IP address from the Name Service using the generated computer name (*pronam*) when the application is started.

If there is no IPv6 address in the object tables for the client, UTM returns blanks in *ip_addr_v6*.

ip_v Specifies whether the IP address used by UTM for this connection is an IPv4 or an IPv6 address:

'V4' IPv4 Address.

'V6' IPv6 address or IPv4 address embedded in IPv6 format.

If no IP address can be returned, openUTM returns blanks.

kc_queue_str - Properties of temporary queues

The *kc_queue_str* data structure is defined for the KC_QUEUE object type. In the case of KC_GET_OBJECT, UTM returns information in *kc_queue_str* about the temporary queues that exist in the application.

Data structure kc_queue_str

```
char qu_name[8];
char qlev[5];
char queue_length[8];
char q_mode;
```

The fields of the data structure have the following meanings:

qu_name
: Name defined or assigned automatically by UTM when the queue was created with QCRE.

qlev
: Contains the maximum number of messages that can be in the queue at any one time.

  openUTM does not take into account the messages created for the queue until the end of the transaction. The number of messages defined in *qlev* for a message queue can therefore be exceeded if several messages were created for the same queue in a single transaction.

  qlev=32767 means there is no limit on the number of messages in the queue.

queue_length
: Contains the number of messages in the queue that are currently being processed or waiting to be processed.

q_mode
: Indicates how UTM responds when the maximum number of messages permitted for the queue is reached. Possible values are:

  ´S´ (STD)
  UTM rejects any further messages for this queue.

  ´W´ (WRAP-AROUND)
  UTM accepts any further messages. When a new message is entered, the oldest message in the queue is deleted.

kc_sfunc_str - Function keys

The data structure r *kc_sfunc_str* is defined for object type KC_SFUNC. In *kc_sfunc_str*, In the case of KC_GET_OBJECT, UTM returns the short description of a function key generated in the application and specifies which function is allocated to this function key.

A transaction code, a command, a KDCS return code can be assigned to a function key or it can be used for the stacking of services.

X/W X/W Function keys are only relevant for UPIC clients on Unix, Linux and Windows systems. in this case, only the parameter *ret* is evaluated.

Data structure kc_sfunc_str

```
char sf_name[4];
char tac[8];
char stack[8];
char ret[3];
char cmd[8];
```

The fields in the data structure have the following meanings:

sf_name

Contains the short description of the function key. Possible values are:

B BS2000 systems: K1 to K14 and F1 to F24

X/W Unix, Linux and Windows systems: F1 to F20

B B B Short messages containing only the value of the key are issued with the K keys. K14 is used for ID card readers (see openUTM manual „Programming Applications with KDCS”, ID card readers).

You can transfer the value of the F key and an input message with the F keys.

tac Contains the name of the transaction code (service TAC) allocated to this function key.

If the function key is pressed when the service is not activated, the service belonging to the transaction code is started.

If the function key is pressed while a service is running, then the function assigned to the function key with *ret* or *stack* takes effect. If these two fields do not contain any values, the first MGET call returns the code 19Z in the next program unit of the service.

stack This is used to stack services. *stack* contains the name of the dialog transaction code assigned to this function key.

If the function key is pressed while a service is active, the current service is stacked and the service with the transaction code in *stack* is started.

If the function key is pressed when no service is active the transaction code contained in the *tac* field is started. If the field *tac* contains no value, pressing the function key causes the service to be started that has the transaction code contained in *stack*.

ret Contains a KDCS return code.

If this function key is pressed while a service is running, then the field KCRCCC in the communication area will contain the return code after the MGET call.

If this key is pressed when a service is started and if *tac* does not contain a value, UTM issues message K009 or starts the BADTACS program unit. This program unit contains the return code assigned to the function key in the first MGET call in the field KCRCCC.

Possible values: 20Z ≤ *ret* ≤ 39Z.

If a UPIC client transmits the function key, on the field *ret* is evaluated.

cmd Name of a KDC command (e.g. KDCOFF or an administration command such as KDCINF) which is activated when the function key is pressed.

If cmd contains a value, the fields *tac*, *ret* and *stack* contain blanks.

X/W

kc_subnet_str - Information on subnets

X/W The data structure *kc_subnet_str* is defined for object type KC_SUBNET. For
X/W KC_GET_OBJECT, UTM returns the generation data for subnets.

X/W **data structure kc_subnet_str**

```
X/W  char mapped_name[8];
X/W  char relevant_bits[3];
X/W  char ip_addr_format[2];
X/W  char ipv4_address[15];
X/W  char ipv6_address[39];
X/W  char bcamappl[8];
```

X/W The fields in the data structure have the following meanings:

X/W mapped_name
X/W Contains the *mapped_name* from the KDCDEF statement SUBNET.

X/W relevant_bits
X/W Contains the number of bits relevant for the subnet mask.

X/W ip_addr_format
X/W Specifies the address format:

X/W V4 The address is an IPv4 subnet address.

X/W V6 The address is an IPv6 subnet address.

X/W ipv4_address
X/W For *ip_addr_format*=V4 contains the IPv4 subnet address, otherwise blanks.

X/W ipv6_address
X/W For *ip_addr_format*=V6 contains the IPv6 subnet address, otherwise blanks.

X/W bcamappl
X/W Contains the BCAMAPPL name from the KDCDEF statement SUBNET.

kc_tac_str - Transaction codes of local services

The data structure *kc_tac_str* is defined for the object type KC_TAC. In the case of KC_GET_OBJECT, UTM returns the following information in *kc_tac_str*:

– properties of a transaction codes or a TAC queue
– statistical information on the load on the service
– the current state of the transaction code or TAC queue

Only the fields *tc_name*, *admin*, *qlev*, *q_mode*, *q_read_acl*, *q_write_acl* and *state* are of any significance to the evaluation of the information of TAC queues (*tac_type*=´Q´).

Transaction codes can be created dynamically with KC_CREATE_OBJECT, deleted with KC_DELETE_OBJECT or modified with KC_MODIFY_OBJECT

	mod[1]	Data structure kc_tac_str
	–	`char tc_name[8];`
	–	`char program[32];`
	x(GPD)	`char lock_code[4];`
	x(GID)	`char state;`
	–	`char tacclass[2];`
	–	`char admin;`
	–	`char call_type;`
	–	`char exit_name[32];`
	–	`char qlev[5];`
	–	`char tac_type;`
	–	`char real_time_sec[5];`
B	–	`char cpu_time_msec[8];`
B	–	`char dbkey[8];`
B	–	`char runprio[3];`
	–	`char api;`
B	–	`char satadm;`
B	–	`char satsel;`
	–	`char tacunit[4];`
B	–	`char tcbentry[8];`
	–	`char in_queue[5];`
	x(GIR)	`char used[10];`
	x(GIR)	`char number_errors[5];`
	x(GIR)	`char db_counter[10];`

mod[1]	Data structure kc_tac_str
x(GIR)	`char tac_elap_msec[10];`
x(GIR)	`char db_elap_msec[10];`
x(GIR)	`char taccpu_msec[10];`
–	`char deleted;`
–	`char pgwt;`
–	`char encryption_level;`
x(GPD)	`char access_list[8];`
–	`char q_mode;`
x(GPD)	`char q_read_acl[8];`
x(GPD)	`char q_write_acl[8];`
–	`char nbr_dputs[10];`
–	`char nbr_ack_jobs[10];`
x(GPD)	`char dead_letter_q;`
x(GIR)	`char nbr_ta_commits[10];`
x(GIR)	`char number_errors_ex[10];`
–	`char in_queue_ex[10];`
x(GIR)	`char taccpu_micro_sec[10];`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 347

The fields in the data structure have the following meanings:

tc_name
: Contains the name of the transaction code or TAC queue whose properties UTM returns. The name is up to 8 characters long.

program
: Contains the name of the program unit assigned to this transaction code.

  For TAC queues, blanks are returned.

lock_code
: Contains the lock code assigned to the transaction code (access protection). Only users/clients who possess the corresponding key code may call this transaction code. The key code must be contained both in the key set assigned to the user ID and in the key set assigned to the LTERM partner via which the user/client is to connect to the application.

The *lock_code* can contain a number between ´0´ and ´4000´.
In KC_CREATE_OBJECT, the maximum value that can be contained by *lock_code* is the maximum value defined using the KEYVALUE operand of the KDCDEF statement MAX.
´0´ means there is no lock code (i.e. the transaction code is not protected by a lock code).

If you want to change the lock code, a key set must not be entered in the *access_list* field.

This parameter is not permitted for TAC queues. In this case, blanks are returned.

state Specifies whether the TAC or the TAC queue is enabled or disabled:

´Y´ TACs: The transaction code is not disabled. It is available after the application has been started. It is available until it is explicitly disabled or deleted.

TAC queues: Writing and reading is permitted.

´N´ TACs: The transaction code is disabled. The lock *state*=´N´ means that UTM will not accept any more jobs for this TAC.
If the transaction code is a KDCS program unit with *call_type*=´B´, then the transaction code is disabled if it is a service TAC (first TAC of a service). If it is a follow-up TAC in a service, however, it is not disabled. Follow-up TACs (*call_type*=´N´) cannot be disabled with *state*=´N´.

TAC queues: Reading is permitted, writing not.

´H´ (HALT)
TACs: The transaction code is completely disabled. The corresponding program unit will not be started anymore by UTM when it is called with this transaction code. For the transaction code of a KDCS program unit, this means that it is disabled if it is a service TAC and if it is a follow-up TAC in an asynchronous or dialog service.
If the transaction code is a service TAC no jobs are accepted for it.
If this TAC is called as a follow-up TAC, then the service is aborted (internal PEND ER with 74Z).

Asynchronous jobs that have already been placed in the message queue of the TAC for temporary storage before the lock are not started. They remain in the queue until the TAC is released or is set to *state*=´N´.

TAC queues: Neither reading nor writing is possible.

´K´ (KEEP)
TACs: This value can only occur in conjunction with asynchronous transaction codes, that are at the same time service TACs (*call_type*=´B´ or ´F´).

UTM accepts jobs for this transaction code. However, instead of being processed, they are simply entered in the job queue for that transaction code. They are processed when you change the status of the transaction code to ´Y´ or ´N´.

TAC queues: writing is permitted, but reading not.

You can disable or release a transaction code or a TAC queue while programs are running.

tacclass
Contains the TAC class assigned to the transaction code. *tacclass* contains a number between 1 and 16 or blanks. The numbers signify:

1 - 8 dialog TAC classes

9 -16 asynchronous TAC classes

If UTM returns blanks in *tacclass*, then the following is true:

– No TAC classes were created during the KDCDEF generation or

– the transaction code in *tc_name* is a dialog TAC (*tac_type*=´D´) that is not assigned to any TAC class.

admin When *tac_type*=´A´ or ´D´, specifies the privileges a user or client requires in order to be able to call this transaction code or a service that contains this transaction code as a follow-up TAC. When *tac_type*=´Q´, *admin* indicates the authorization a user or client needs in order to access this TAC queue.

´Y´ TACs: This transaction code can only be called by a user who has administration privileges.

TAC queues: Only a user with administration privileges can write messages to and read messages from this queue.

´N´ No administration privileges are required for this TAC or TAC queue.

´R´ (READ)
No administration privileges are required for this TAC or TAC queue.

The program unit belonging to the transaction code can use all the functions of KDCADMI that read the application data.

In addition, the access rights to the TAC (*tac_type*=´A´ or ´D´) can be limited by means of a lock code or an access list. If it is a TAC queue (*tac_type*=´Q´), it is possible to restrict the access rights by means of the parameters *q_read_acl* and/or *q_write_acl*.

call_type
: Specifies if a service (for example the first TAC of a service) is being started with the transaction code or if the transaction code is a follow-up TAC in a service.

  ´B´ (BOTH)
  A service can be started with the TAC. The TAC can also be a follow-up TAC in a service, however.

  ´F´ (FIRST)
  A service can be started with the transaction code.

  ´N´ (NEXT)
  The transaction code can only be a follow-up TAC in a service. It can only be disabled with *state*=´H´.

exit_name
: Contains the name of the event exit VORGANG assigned to this TAC.

qlev (**q**ueue **lev**el)
: For asynchronous transaction codes (*tac_type*=´A´) or queues (*tac_type*=´Q´), *qlev* specifies the maximum number of messages allowed in the message queue for this transaction code or in the TAC queues. If this control value is exceeded, how openUTM responds depends on the value in the *q_mode* field.

  UTM does not take into account the messages created for the queue until the end of the transaction. The number of messages specified for a message queue in *qlev* can therefore be exceeded if several messages are created for the same queue in a single transaction.

tac_type
: Specifies if jobs sent to this transaction code will be processed in the dialog or asynchronously or whether a TAC queue was generated.

  ´D´ This transaction code is a dialog TAC, i.e. jobs sent to this transaction code will be processed in the dialog with the job-submitter.

  ´A´ This transaction code is an asynchronous transaction code. When calling this transaction code, an asynchronous job is created that is temporarily stored in the message queue of the transaction code. The job is processed separately from the job-submitter.

  ´Q´ A TAC queue was generated.

  A message can be written to such a queue with a DPUT call and read from the queue with a DGET call.

real_time_sec

Contains the maximum amount of real time in seconds that a program unit may use if it is started via this transaction code. If the program unit runs longer, then UTM aborts the service and outputs a UTM message.

real_time_sec=´0´ means that the real time used by the program unit will not be monitored.

B cpu_time_msec

B Contains the maximum CPU time in milliseconds that the program unit with this
B transaction code may use while processing. If the program unit runs longer, then
B UTM aborts the service and outputs a UTM message.
B The value ´0´ means that the time will not be monitored for the program unit started
B via this transaction code.

B dbkey *dbkey* is only relevant if the program unit belonging to the transaction code makes
B database calls and if the database system is linked to UTM.
B *dbkey* contains the database key that UTM passes from the program unit to the
B database system in a database call. The format of the key depends on the database
B system used. The key is a maximum of 8 characters long.
B *dbkey* is only supported for UDS at the present.

B The value *dbkey*='UTM' will result in the value of the start parameter DBKEY being
B passed to the database (see the openUTM manual “Using openUTM Applications
B on BS2000 Systems”; start parameters).

B runprio

B Contains the run priority setting of the BS2000 system for the transaction code. This
B run priority will be assigned to the UTM process in which the corresponding
B program unit runs. In this manner you can utilize the scheduling mechanism of the
B BS2000 system for run-time control of UTM program unit runs. The run priority does
B not have any influence, however, on the point in time at which UTM starts a program
B unit.

B When a program unit is started, UTM attempts to set the run priority of the current
B process to the value in *runprio*. If the run priority generated is not compatible with
B the JOIN entries of the corresponding user ID, then the run priority of the current
B process is not changed. UTM outputs the corresponding K message. If the
B maximum number of *runprio* values allowed for the user ID and the job class are
B different, then the value most favorable for the user is allowed to be used. If there
B are no JOIN entries, then the run priority specified in *runprio* is used.

B After the end of a program unit run, UTM sets the run priority back to the original
B value unless the run priority was changed again during the program unit run with
B the CHANGE TASK PRIORITY command. In this case, the run priority that was set
B externally will be maintained after the end of the program unit run.

B If *runprio*=´0´, then no specific run priority is generated for this transaction code.

api (**a**pplication **p**rogramming **i**nterface)
Specifies which programming interface is used by the program unit belonging to the transaction code.

´K´ KDCS

´C´ CPI-C

´X´ XATMI

B satadm
B Specifies if UTM SAT administration privileges are required to call the transaction
B code.

B ´Y´ The TAC may only be called by users, clients or partner applications that are
B permitted to carry out administration operations on the SAT logging within
B the application (UTM SAT administration privileges).

B ´N´ The transaction code may also be called by users, clients and partner appli-
B cations that do not have UTM SAT administration privileges.

B satsel Specifies which events SAT will log during the corresponding program unit run
B (TAC-specific setting). One requirement for logging is that SAT logging is enabled
B for the application (*kc_max_par_str.sat*=´Y´). See also the openUTM manual “Gener-
B ating Applications” and openUTM manual “Using openUTM Applications on
B BS2000 Systems” for more information on SAT logging.

B ´B´ (BOTH)
B Both successful **and** unsuccessful events are logged.

B ´S´ (SUCCESS)
B Only successful events are logged.

B ´F´ (FAIL)
B Only unsuccessful events are logged.

B ´N´ (NONE)
B No TAC-specific type of SAT logging is defined.

tacunit Contains the number of accounting units charged for each call of the transaction code in the accounting phase of UTM Accounting.
The accounting units are added to the accounting unit counter of the user ID that called the transaction code.

B tcbentry
B Contains the name of the KDCDEF control statement TCBENTRY in which the TCB
B entries assigned to this TAC are collected.

in_queue

Only contains data for asynchronous TACs.
Specifies how many asynchronous messages are temporarily stored in the message queue of the transaction code that must still be processed by the corresponding program unit.

If this number of messages is greater than 99999, then the number is not displayed in full. You should therefore use the field *in_queue_ex* (see page 558) since larger numbers can be entered in full here.

used Specifies the number of program unit runs processed in all with this transaction code since the *used* counter was last reset.

You can reset the counter to 0 using KC_MODIFY_OBJECT.
In UTM-S applications *used* is automatically rest to 0 only in regenerations with KDCDEF and in each update generation with KDCDEF/KDCUPD. In UTM-F applications the *used* counter is automatically reset to 0 when the application is started.

number_errors

Specifies how many of the program unit runs started with this transaction code terminated with errors since the *number_errors* counter was last reset to 0.

You can reset the counter to 0 using KC_MODIFY_OBJECT.
In UTM-S applications *number_errors* is automatically rest to 0 only in regenerations with KDCDEF and in each update generation with KDCDEF/KDCUPD. In UTM-F applications the *number_errors* counter is automatically reset to 0 when the application is started.

If the number of program unit runs is greater than 99999, then the number is not displayed in full. You should therefore use the field *number_errors_ex* (see page 558) since larger numbers can be entered in full here.

db_counter

Contains the average number of database calls from a program unit started using this transaction code since the *db_counter* counter was last reset to binary 0.

db_counter is always 0 for database link via the XA interface.
You can reset the counter to 0 using KC_MODIFY_OBJECT.

tac_elap_msec

Contains the average runtime of the program units started using this transaction code since the *tac_elap_msec* counter was last reset (elapsed time); specified in milliseconds. You can reset the counter to 0 using KC_MODIFY_OBJECT.

db_elap_msec

Contains the average time needed for processing database calls in program unit runs using this TAC; specified in milliseconds. *db_elap_msec* considers all database calls made since the counter was last reset.

db_elap_msec is always binary 0 for database link via the XA interface.
You can reset the counter to 0 using KC_MODIFY_OBJECT.

taccpu_msec

Contains the average CPU time in milliseconds needed to process this transaction code in the program unit. The value corresponds to the CPU time needed by UTM plus the CPU time used by the database system; specified in milliseconds. *taccpu_msec* considers all program unit runs since the counter was last reset to 0.
You can reset the counter to 0 using KC_MODIFY_OBJECT.

deleted

Specifies whether or not the transaction code or the TAC queue was deleted from the configuration.

´Y´ The transaction code or the TAC queue was deleted but the name is disabled. You cannot generate a new transaction code or a new TAC queue with this name.

´N´ The transaction code or the TAC queue was not deleted.

pgwt Only contains a value if your application processes jobs to the TAC classes using priority control, i.e. only if the KDCDEF generation contains the TAC-PRIORITIES statement.

pgwt specifies, whether blocking calls (e.g. PGWT) can be processed in a program unit run started for this transaction.

´Y´ Blocking calls are allowed.

´N´ Blocking calls are not allowed.

encryption_level

Only relevant for service TACs (*call_type*=´F´or ´B´)
encryption_level specifies, whether messages for this transaction code must be encrypted or not.

´N´ (NONE)
Message encryption is not required. A client can start a service using this transaction code, even if the client does not encrypt the input message. The output message to the client is only encrypted if the relevant input message from the client was encrypted also.

´1´ (Level 1)
The input message has to be encrypted using the DES algorithm in order to access this transaction code.

´2´ (Level 2)
The input message has to be encrypted using the AES algorithm in order to access this transaction code.

If *encryption_level* =´1´ or ´2´ is specified, a client can only start a service through this transaction code if the client meets one of the following prerequisites:

– The client is a “trusted” client (see *kc_pterm_str* or *kc_tpool_str* field *encryption_level*). A “trusted” client can start a service through the transaction code, even if the input message is not encrypted.

– The client has encrypted the input message to the transaction code with at least the specified encryption level. If a “not trusted” client does not encrypt the first input message or does not encrypt it to the required level or if the client does not support encryption, no service is started.

All output messages to a not trusted client are encrypted.

If the transaction code is started using service concatenation, the first input message from the client does not need to be encrypted.

If the transaction code is called without user data or if it is started through service concatenation, then the client must be able to encrypt data. openUTM encrypts all output dialog messages to the client and expects all consequent input messages from a not trusted client to be encrypted in multistep services.

access_list
: Contains the name of a key set that describes the access rights of users to this transaction code.

  It is not permitted to specify *access_list* with TAC queues.

  access_list and *lock_code* must not have the same values.

  A user can only access the transaction code when the key set of the user, the key set of the LTERM partner by means of which the user is signed on and the key set specified by means of *access_list* contain at least one key code in common.

  You can remove data access control by filling *access_list* with blanks.

  If neither *access_list* nor *lock_code* contains a value, any user can access the transaction code.

q_mode (queue mode)
: Defines how openUTM responds when a queue already contains the maximum number of messages and the queue level has thus been reached.

  ´S´ UTM rejects any further jobs.

  ´W´ (only when *tac_type*=´Q´)
  : UTM accepts further jobs but deletes the oldest messages in the queue.

q_read_acl (only when *tac_type*=´Q´)
: Indicates the rights (name of a key set) required by a user in order to be able to read and delete messages from this queue.

  A user can only have read access to this TAC queue if the key set of the user and the key set of the logical terminal by means of which the user is signed on each contain at least one key code that is also contained in the displayed key set.

  If *q_read_acl* does not contain a value, all users can read and delete messages from this queue.

q_write_acl (only when *tac_type*=´Q´)
: Indicates the rights (name of a key set) required by a user in order to be able to write messages to this queue.

  A user can only have write access to this TAC queue if the key set of the user and the key set of the logical terminal by means of which the user is signed on each contain at least one key code that is also contained in the displayed key set.

  If *q_write_acl* does not contain a value, all users can write messages to this queue.

nbr_dputs
: Number of pending time-controlled jobs for this TAC whose start point has not yet been reached.

nbr_ack_jobs
: Number of pending acknowledgment jobs for this TAC that have not yet been activated.

dead_letter_q
: Specifies whether a queued message should be retained in the dead letter queue if it was not processed correctly and it has not been redelivered.

  'Y' Errored queued messages are backed up in the dead letter queue.

  dead_letter_q ='Y' is not permitted for KDCDLETQ, KDCMSGTC, all interactive TACs and asynchronous TACs with CALL=NEXT.

  'N' Errored queued messages are deleted if they are not redelivered.

nbr_ta_commits
: Number of program unit runs for this TAC which have successfully completed a transaction.

  You can reset the counter to 0 using KC_MODIFY_OBJECT.

number_errors_ex
: See *number_errors* on page 554.

in_queue_ex
: See *in_queue* on page 554.

taccpu_micro_sec
: Contains the average CPU time in microseconds taken to process this transaction code in the program unit. This corresponds to the CPU time consumed by UTM plus the CPU time required by the database system.
  taccpu_micro_sec takes account of all program runs since the counter was last reset. You can use KC_MODIFY_OBJECT to reset the counter to 0.

kc_tacclass_str - TAC classes for the application

The data structure *kc_tacclass_str* is defined for the object type KC_TACCLASS. In the case of KC_GET_OBJECT, UTM returns the following information in *kc_tacclass_str*:

- properties of the TAC class
- statistical information on how often and for how long jobs for the TAC class had to wait for processing
- the current maximum number of processes that may simultaneously process jobs for the transaction code of the TAC class if the application was generated without priority control (i.e. without the TAC-PRIORITIES statement).

mod[1]	Data structure kc_tacclass_str
–	`char tacclass[2];`
x(A)[2]	`char tasks[3];`
x(A)[2]	`char tasks_free[3];`
–	`char pgwt;`
–	`char waiting_msgs[10];`
x(GIR)	`char avg_wait_time_msec[10];`
–	`char prio[3];`
x(GIR)	`nr_calls[10];`
x(GIR)	`nr_waits[10];`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 351

[2] These properties can only by modified if the application was generated without the TAC-PROPERTIES statement. Only one of these fields may be specified in a KC_MODIFY_OBJECT call.

The fields in the data structure have the following meanings:

tacclass

Contains the number of the TAC class. A number between 1 and 16 is output for *tacclass*.

The TAC classes from 1 to 8 are dialog TAC classes.
The TAC classes from 9 to 16 are asynchronous TAC classes.

tasks Only relevant if priority control was not generated for the TAC class (KDCDEF generation without TAC-PRIORITIES statement).

Specifies how many processes of the application may process TACs of the TAC class *tacclass* at the same time (absolute number).

See also page 353.

If the application is generated with priority control, *tasks* contains a blank.

tasks_free
Only relevant if the application was generated without the TAC-PRIORITIES statement.

For dialog TAC classes *tasks_free* contains the minimum number of processes of the application that must be kept free for processing transaction codes from other TAC classes. For asynchronous TAC classes *tasks_free* contains the minimum number of processes that must be kept free for processing transaction codes from other asynchronous TAC classes.

UTM returns ´0´ to *tasks_free* if the value of *tasks_free* was defined neither during KDCDEF generation nor by means of administration functions, or if a value was defined for *tasks* the last time the number of processes for the TAC class was modified.

See also page 353.

If the application is generated with priority control, *tasks_free* contains blanks.

pgwt Specifies if program units that contain blocking calls, for example the KDCS call PGWT, are allowed to run in this TAC class.

´Y´ Blocking calls are allowed in this TAC class.

´N´ Blocking calls are not allowed in this TAC class.

Program units containing blocking calls are allowed in at most one dialog TAC class and one asynchronous TAC class.

waiting_msgs
Contains the number of jobs for transaction codes of this TAC class that are currently in temporary storage in UTM and that have not yet been processed.

avg_wait_time_msec
Contains the average wait time of jobs in the job queue assigned to the transaction code of this TAC class.

If there is no process for the TAC class, UTM accepts jobs for the TAC class (using free processes that are not “allowed” to process jobs to this TAC class) and temporarily stores them in the KDCFILE. This is always the case when there are jobs for

TAC classes with a higher priority level (with priority control) or (in the case of process restriction) if the maximum number of processes that the TAC class is allowed to process has already been reached (see *tasks*, *tasks_free*).

The time between accepting a job and starting to process it is the wait time displayed here.

The value for *avg_wait_time_msec* is in milliseconds.

The value of *avg_wait_time_msec* can be reset to 0. If this value is reset then the values of *nr_calls* and *nr_waits* is also implicitly reset.

prio Contains the type of priority control generated for this TAC class.
The following values are possible:

´ABS´ Absolute priorities:
A free process is always assigned to the TAC class with the highest priority, i.e. priority 1 to 9, if jobs are waiting.The TAC class with the next lowest priority is not served until there are no more jobs with the higher priority level waiting in the TAC class.

´REL´ Relative priorities:
Free processes are more frequently allocated to higher TAC classes than to lower TAC classes if jobs are waiting to be processed.

´EQ´ Equal priorities:
If there are any jobs waiting, all TAC classes are served at an equal rate.

´NO´ No priority control was generated.

nr_calls
Number of program unit runs for this TAC class.

You can reset the value to 0 using KC_MODIFY_OBJECT. If this value is reset then the values *avg_wait_time_msec* and *nr_waits* are also implicitly reset.

nr_waits
Number of wait situations taken into account to calculate the value *avg_wait_time_msec*.

You can reset the value to 0 using KC_MODIFY_OBJECT. If this value is reset then the values *avg_wait_time_msec* and *nr_calls* are also implicitly reset.

kc_tpool_str - LTERM pools for the application

The data structure *kc_tpool_str* is defined for the object type KC_TPOOL. In the case of KC_GET_OBJECT, UTM returns the following information on an LTERM pool in *kc_tpool_str*:

- the number of LTERM partners currently permitted for the LTERM pool
- the properties of the LTERM partners of the LTERM pool
- the type of clients that may connect to the application via this LTERM pool
- statistical data on the workload of the LTERM pool.

	mod[1]	Data structure kc_tpool_str
	–	`char lterm[8];`
	–	`char pronam[8];`
	–	`char ptype[8];`
	–	`char bcamappl[8];`
	–	`char connect_mode;`
	–	`char max_number[10];`
	–	`char kset[8];`
B	–	`char locale_lang_id[2];`
B	–	`char locale_terr_id[2];`
B	–	`char locale_ccsname[8];`
	–	`char lock_code[4];`
	x(GP)[2]	`char state;`
	x(GP)[2]	`char state_number[10];`
B	–	`char format_attr;`
B	–	`char format_name[7];`
	–	`char qlev[5];`
	–	`char termn[2];`
B	–	`char annoamsg;`
B	–	`char netprio;`
B	–	`char protocol;`
	–	`char actcon[10];`
	–	`char maxcon[10];`
	–	`char map;`
	x(GP)	`char idletime[5];`

mod[1]	Data structure kc_tpool_str
–	`char encryption_level;`
–	`char user_kset[8];`
–	`char usp_hdr;`
–	`char kerberos_dialog;`

B

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 355.

[2] Wth KC_MODIFY_OBJECT both fields must be specified together.

The fields in the data structure have the following meanings:

lterm Contains the prefix for the names of the LTERM partners of the LTERM pools. The names of the LTERM partners consists of this prefix and a sequential number. The sequence goes from 1 up to the value returned in *max_number*.

Example

If *max_number*=´1000´ and *lterm*=´LTRM´, then the LTERM partners of the LTERM pool are named LTRM0001, LTRM0002, ..., LTRM1000.

pronam

Specifies the computer on which the clients must be located in order to connect to the application via this LTERM pool.

UTM returns either the symbolic name under which the computer is known to the local transport system or the value ´*ANY´ for an open LTERM pool.

´*ANY´ means:
Every client can sign on to the application via the LTERM pool if the client fulfills the following conditions:

– Its terminal type matches the type specified in *ptype*.

– It was not explicitly added to the configuration (with the KDCDEF statement PTERM or dynamically with object type KC_PTERM).

– No other LTERM pool exists for the computer on which the client resides nor for its terminal type (*ptype*).

ptype The type of clients that are allowed to connect to the application via this LTERM pool. You can determine the meaning of the value returned by UTM in *ptype* from the table on page 531 (BS2000 systems) or from the table on page 534 (Unix, Linux and Windows systems).

If *ptype*=´*ANY´, then it is an open LTERM pool. All clients resident on or connected to the computer specified in *pronam* and for which the following statements are true can connect via this LTERM pool:

– The client is not entered explicitly in the configuration.

– No LTERM pool exists for which the client type is set in *ptype* for the computer in *pronam*.

bcamappl
The name of the local UTM application (BCAMAPPL name) via which the connection between the client and the UTM application will be established.

This name must be specified by the client when it wants to establish a connection to the local application.

connect_mode
Specifies if a client can connect to the UTM application via the LTERM pool more than once under the same name.

´S´ Each client can only connect once under the same name via the LTERM pool.

´M´ An UPIC client (*ptype*=´UPIC-R´ or ´UPIC-L´) or a TS application (='APPLI' or ´SOCKET´) that runs more than once on the same computer can connect to the UTM application via the LTERM pool more than once under the same name. A new name does not have to be created for every connection.

The UPIC client or the TS application can connect to the LTERM pool as many times as there are LTERM partners allowed for the LTERM pool. The name of the corresponding pool LTERM partner will be set in this case to the name of the client or TS application, i.e. the partner will then be identified in the application by the name triplet (name of the LTERM partner, *pronam* and *bcamappl)*. The UPIC client or the TS application is not known in the UTM application under its local name or its application name.

max_number
Specifies the maximum number of clients that may be simultaneously connected via this LTERM pool, i.e. *max_number* specifies how many LTERM partners comprise this LTERM pool.

kset
: Contains the name of the key set assigned to the LTERM pool. The key set determines which transaction codes the clients that connect to the application via this LTERM pool may call. The clients may only start a transaction code if the key set contains a key code that numerically matches the lock code of the transaction code, or if the transaction code does not have access security, i.e. it does not possess a lock code.

  If the LTERM pool is not assigned a key set, then *kset* contains blanks.

  The following applies for *ptype*=´UPIC-...´, ´APPLI´ or ´SOCKET´:

  kset specifies the maximum number of access of a client which connects through this LTERM pool.

  kset always comes into effect when the client passes a true user ID to UTM during session/conversation establishment. The access privileges result from the set of key codes contained both in the key set of the user ID an in *kset*.

  If the client does not pass a true user ID to openUTM for the session/conversation, the access privileges result from the subset of key codes in *kset* an *user_kset* (minimum access rights).

B locale_lang_id, locale_terr_id, locale_ccsname
: These contain the three components of the locale assigned to the LTERM pool. The locale defines the language environment of the clients that connect to the application via this LTERM pool (see also the openUTM manual “Generating Applications”).

B *locale_lang_id*
: Contains the up to two characters long language code.

B *locale_terr_id*
: Contains an up to two characters long territory code.

B *locale_ccsname*
: (**c**oded **c**haracter **s**et **name**)
  Contains the name (up to 8 characters) of an expanded character set (CCS name; see also the XHCS User Guide).

lock_code
: Contains the lock code assigned to the LTERM partners of the LTERM pool (access protection). Only users/clients who possess the corresponding key code may connect via this LTERM pool.
  The *lock_code* can contain a number between ´0´ and ´4000´. ´0´ means that the LTERM pool is not protected by a lock code.

state, state_number

The number of LTERM partners comprising this LTERM pool is set in the KDCDEF generation of the LTERM pool (see *max_number*). The number of LTERM partners via which clients can connect to the application can, however, be reset to a smaller value during operation by the administration. The rest of the LTERM partners are disabled by this action. In the *state* and *state_number* fields UTM specifies how many LTERM partners of the LTERM pool are currently permitted, i.e. not disabled. The number of LTERM partners allowed determines how many clients can connect to the application via this LTERM pool at the same time.

If *state* contains the value 'Y', the pool is permitted for the number of communication partners specified in *state_number* (ON). If *state* contains the value 'N', the pool is locked for the number of communication partners specified in *state_number* (OFF).

If all LTERM partners of the LTERM pool are disabled, then *state* contains the value ´Y´ and *state_number* the value ´0´.

B format_attr, format_name

B These define the start format for users on terminals connected via this LTERM pool. After the connection between the terminal and the application is established, the formats described in *format_attr* and *format_name* will be output on the terminal as long as no terminal-specific restart is being executed.

B *format_attr*

B Contains the format code:

B ´A´ (format attribute ATTR)
The start format is a format with user attributes. The properties of the format fields can be changed by the KDCS program unit. The format name at the KDCS program interface is +*format_name*.

B ´N´ (format attribute NOATTR)
The start format is a format without user attributes. Neither the field nor the format properties can be changed by the KDCS program units. The format name at the KDCS program interface is **format_name*.

B ´E´ (format attribute EXTEND)
The start format is a format with expanded user attributes. The properties of the format fields as well as global format properties can be changed by the KDCS program unit. The format name at the KDCS program interface is #*format_name*.

B *format_name*

B Contains the name of the start format. The name can be up to 7 characters long and B contains only alphanumeric characters.

qlev (**q**ueue **lev**el)
Specifies the maximum number of asynchronous messages that may be temporarily stored in the message queue of the LTERM partner belonging to this LTERM pool for processing at one time by UTM. If the control value for an LTERM partner of the LTERM pool is exceeded, then UTM will reject any additional asynchronous jobs sent to this LTERM partner. The control value is specified in the KDCDEF generation.

termn (**ter**minal **mn**emonic)
Contains the code for the type of client that can connect via this LTERM pool. When running, UTM KDCS program units that were started via the LTERM pool provide the code in the KCTERMN field of the communication area header. The code is a maximum of 2 characters long. The values that *termn* may contain can be obtained from the table for *ptype* on page 531 (BS2000 systems) or on page 534 (Unix, Linux and Windows systems).

B annoamsg (**anno**unce **a**synchronous **m**e**s**sa**g**e)
Specifies if UTM will announce asynchronous messages on the terminal with a UTM message in the system line before output.

´Y´ UTM announces every asynchronous message to this terminal with the UTM message K012 in the system line. The user must then explicitly request the asynchronous message with the KDCOUT command.

´N´ Asynchronous messages are output on the terminal immediately, i.e. without announcement. For *annoamsg* = ´N´, the establishing of the connection to this LTERM pool via a multiplex connection will only be possible starting with OMNIS V7.0.

B netprio Specifies the transport priority used on the transport connection between the application and the clients connected via this LTERM pool.

´M´ Medium transport priority

´L´ Low transport priority

protocol

Specifies whether the NEABT user service protocol will be used on connections between the UTM application and a client that connects via this LTERM pool.

´N´ The user protocol service will not be used between the UTM application and the client/printer.

For UPIC clients (*ptype*=UPIC-R) and TS applications (*ptype*=APPLI or SOCKET), *protocol*=´N´ will always be output.

No connections can be established via a multiplex connection to an LTERM pool for which *protocol*=´N´ is set.

´S´ (STATION)
The user protocol service (NEABT) is used between the UTM application and the client/printer. UTM uses the NEABT user protocol service for LTERM pools with *ptype*=*ANY, for example to determine the type (*ptype*) of a client. In this case, NEABT is always used.

actcon

Specifies how many clients are currently connected to the application via this LTERM pool.

maxcon

Contains the maximum number of clients that were simultaneously connected to the application via this LTERM pool in the current application run.
The counter is reset to 0 at the start of the application.

map Specifies whether UTM converts the user messages before sending them to a partner or after receiving them from a partner.

´U´ (USER)
UTM does not convert the data. The message is passed to the partner application unchanged.

´S´, ´1´, ´2´, ´3´, ´4´
Only possible if communication to the TS application is realized via the socket interface (SOCKET).

The data pertaining to a message is converted from EBCDIC to ASCII before being sent and from ASCII to EBCDIC after being received. In this context, UTM assumes that the message contains printable characters only.

With ´S´ (SYSTEM) and ´1´, data is converted using the standard code tables, i.e. from EBCDIC to 7-bit ASCII. ´2´, ´3´ and ´4´ stand for the three conversion tables that you can define yourself (see the openUTM manual “Generating Applications”, keyword „code conversion tables“).

X/W ´S´ (SYSTEM)
X/W is allowed on Unix, Linux and Windows systems only if the client is a TS
X/W application (*ptype*=´APPLI´ or ´SOCKET´).

X/W With ´S´, UTM converts the data in a message being passed between a
X/W partner application and a local application. Before a message is sent, the
X/W data is converted from ASCII to EBCDIC, and after it has been received it
X/W is converted from EBCDIC to ASCII. In this context, UTM assumes that the
X/W message contains printable characters only.

X/W On Unix, Linux and Windows systems, UTM never performs message
X/W handling for formatted messages.

idletime

idletime contains the time in seconds which UTM waits for a response from a client after a single-step transaction is terminated or after sign-off (KDCSIGN). If the time is exceeded, the connection to the client is closed down. If the client is a terminal, message K021 was issued before connection shutdown.

The value 0 means wait without time limit.

encryption_level

Only relevant for UPIC clients and, on BS2000 systems, for some terminal emulations.
encryption_level specifies whether, on the connection to the client that wants to connect to the application via the LTERM pool, the UTM application

– wants to demand encryption of messages by default,
– if it does, which encryption level must be used,
– wants to know whether the clients are „trusted“ clients.

The following values are possible:

´N´ (NONE)
UTM does **not** want the messages to be encrypted by default.
Services for which encryption was generated (see *kc_tac_str.encryption_level* on page 556) can only be started by a client connected through this pool if the client agrees encryption when setting up the connection.

´1´ (LEVEL 1)
UTM demands by default the encryption of messages with encryption level 1. In other words, the messages are encrypted with the DES algorithm and an RSA key with a key length of 200 bits is used for exchange of the DES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´2´ (LEVEL 2)
UTM demands by default the encryption of messages with encryption level 2. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of512 bits is used for exchange of the AES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´3´ (LEVEL 3)
UTM demands by default the encryption of messages with encryption level 3. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 1024 bits is used for exchange of the AES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´4´ (LEVEL 4)
UTM demands by default the encryption of messages with encryption level 4. In other words, the messages are encrypted with the AES algorithm and an RSA key with a key length of 2048 bits is used for exchange of the AES key.
Connection establishment to the client is rejected by UTM if the client does not support at least this encryption level.

´T´ (TRUSTED)
The client is a “trusted” client. Messages exchanged between the client and the application are not encrypted. A “trusted” client can also start services for which the service TAC requires encryption (generated with *kc_tac_str.encryption_level* =´1´ or ´2´; see page 556).

user_kset

Only relevant with *ptype*=´UPIC-...´, ´APPLI´ or ´SOCKET´.

user_kset contains the name of the key set defining the minimum access privileges of the client in the local application.

The key set specified in *user_kset* only comes into effect if the client has signed on under the connection user ID (see also *kset*).

The access rights in *kset* always apply.

usp_hdr

Indicates the output messages for which UTM creates a UTM socket protocol header on this connection. Possible values are:

´A´ (ALL)
UTM creates a socket protocol header for all output messages (dialog, asynchronous, K messages) and precedes the message with it (ALL).

´M´ (MSG)
UTM creates a UTM socket protocol header for the output of K messages and precedes the message with it (MSG).

´N´ (NO)
UTM does not create a UTM socket protocol header for any output message (NO).

The values ´A´ and ´M´ can only occur for LTERM pools that are configured for communication via socket connections (*ptype*=SOCKET).

B kerberos_dialog

B ´Y´ When the connection is established ,a Kerberos dialog is conducted for clients that support Kerberos and are connected directly to the application via this terminal pool (not via OMNIS).

B ´N´ No Kerberos dialog is performed.

B For more detailed information, refer to the openUTM manual “Generating Applications”.

kc_transfer_syntax_str - Transfer syntax for communication via OSI TP

The data structure *kc_transfer_syntax_str* is defined for object type KC_TRANSFER_SYNTAX. In the case of KC_GET_OBJECT, UTM returns the local name and the object identifier of a transfer syntax in *kc_transfer_syntax_str*.
During communication via OSI TP the transfer syntax specifies in which form the user data is transferred to the communication partner. Both communication partners must use the same transfer syntax on a connection.

Data structure kc_transfer_syntax_str

```
char transfer_syntax_name[8];
char object_id[10][8];
```

The fields of the data structure have the following meanings:

transfer_syntax_name
: Contains the name generated locally for the transfer syntax. It is at most 8 characters long.

object_id
: Contains the object identifier of the transfer syntax.

  The object identifier consists of at least 2 and at most 10 components. The individual components are positive integers between 0 and 67108863.

  For each component of the object identifier, UTM returns a field element, i.e. the number of occupied field elements in *object_id* corresponds to the number of components. The remaining field elements contain binary zeros.

  For further information on the object identifier see the openUTM manual “Generating Applications”.

kc_user_str, kc_user_fix_str, kc_user_dyn1_str and kc_user_dyn2_str user IDs

The data structures *kc_user_str*, *kc_user_fix_str*, *kc_user_dyn1_str* and *kc_user_dyn2_str* are defined for the object types KC_USER, KC_USER_FIX, KC_USER_DYN1 and KC_USER_DYN2. The data structure *kc_user_str* is subdivided into three substructures in order to improve performance when accessing user data in UTM cluster applications. All the data in UTM cluster applications stored in the cluster user file is located in the data structure *kc_user_dyn2_str*.

User IDs can be dynamically created with KC_CREATE_OBJECT, deleted with KC_DELETE_OBJECT or modified with KC_MODIFY_OBJECT.

If you want to create user IDs or make modifications, you must use the structure *kc_user_str*. The other structures are only intended for read operations with KC_GET_OBJECT.

In the case of KC_GET_OBJECT, UTM returns the following information concerning the user ID in *kc_user_str, kc_user_fix_str, kc_user_dyn1_str* and *kc_user_dyn2_str,* :

- The attributes assigned to this user ID, such as the type and method of authentication (password, magnetic stripe card), start format, access privileges, administration privileges.
- The number of jobs entered by this user ID and statistical data on the resources demanded while processing the jobs.
- The number of asynchronous jobs running under this user ID.
- The number of users currently signed on with the application under this user ID and the time of the last sign-on under this user ID.
- The number of security violations by users/clients that have signed on using this user ID.
- The properties of the associated USER queue

	mod[1]	Data structure kc_user_str	Page[2]
	–	`char us_name[8];`	578
	x(GPD)	`char kset[8];`	578
	x(GPD)	`char state;`	578
B	–	`char card_position[3];`	578
B	–	`char card_string_lth[3];`	578
B	–	`char card_string_type;`	578
B	–	`union kc_string card_string;`	578
	x(GPD)	`union kc_pw password;`	579
	x(GPD)	`char password_type;`	580
	–	`char password_dark;`	580
B	–	`char card_id[32];`[3]	580
B	x(GPD)[4]	`char format_attr;`	580
B	x(GPD)[3]	`char format_name[7];`	580
B	–	`char locale_lang_id[2];`	581
B	–	`char locale_terr_id[2];`	581
B	–	`char locale_ccsname[8];`	581
	–	`char protect_pw_lth;`	581
	–	`char protect_pw_compl;`	582
	–	`char protect_pw_time[3];`	582
	–	`char restart;`	582
	–	`char permit;`	583
B	–	`char satsel;`	583
	–	`char user_type;`	584
	–	`char lterm_curr[8];`	584
	–	`char connect_mode;`	584
	–	`char in_service;`	585
	–	`char number_tacs[10];`	585
	–	`char cputime_sec[10];`	585
	–	`char seccounter[5];`	585
	–	`char deleted;`	585
	x	`char protect_pw_time_left[3];`	585
	–	`union kc_sign_date sign_time_date;`	586

	mod[1]	Data structure kc_user_str	Page[2]
	–	`char asyn_services[10];`	586
	–	`char clients_signed[10];`	587
	–	`char protect_pw_min_time[3];`	587
	–	`char qlev[5];`	587
	–	`char out_queue[5];`	587
	x(GPD)	`char q_read_acl[8];`	587
	x(GPD)	`char q_write_acl[8];`	588
	–	`char q_mode;`	588
B	–	`char certificate[10];`	588
B	–	`char cert_auth[10];`	588
	x(GIR)	`char bcam_trace;`	588
B	–	`char principal[100];`	589
	–	`char node_last_excl_signon[4]`	589
	–	`char exclusively_signed;`	589
	–	`union kc_sign_date excl_sign_time_date;`	589
	–	`char out_queue_ex[10];`	590
	–	`char ptc;`	590
	–	`char bound_ptc;`	590
	–	`char bound_service;`	590
	--	`char cputime_msec[10];`	590

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 357.

[2] The meaning of the fields is described on the pages indicated in this column..

[3] By default, this is filled with blanks.

[4] When you change the start format with KC_MODIFY_OBJECT you must enter values for *format_name* **and** *format_attr*.

	Data structure kc_user_fix_str	Page[1]
	`char us_name[8];`	578
B	`char card_position[3];`	578
B	`char card_string_lth[3];`	578
B	`char card_string_type;`	578
B	`union kc_string card_string;`	578
B	`char card_id[32];`[2]	580
	`char restart;`	582
	`char permit;`	583
B	`char satsel;`	583
	`char user_type;`	584
	`char qlev[5];`	587
B	`char certificate[10];`	588
B	`char cert_auth[10];`	588
B	`char principal[100];`	589

[1] The meaning of the fields is described on the pages indicated in this column..

[2] By default, this is filled with blanks.

	Data structure kc_user_dyn1_str	Page[1]
	`char us_name[8];`	578
	`char kset[8];`	578
	`char state;`	578
B	`char format_attr;`	580
B	`char format_name[7];`	580
	`char lterm_curr[8];`	584
	`char connect_mode;`	584
	`char in_service;`	585
	`char number_tacs[10];`	585
	`char cputime_sec[10];`	585
	`char asyn_services[10];`	586
	`char deleted;`	585
	`char out_queue[10];`	587

Data structure kc_user_dyn1_str	Page[1]
char q_read_acl[8];	587
char q_write_acl[8];	588
char q_mode;	588
char bcam_trace;	588
char clients_signed[10];	587
union kc_sign_date sign_time_date	586
char cputime_msec[10];	590

[1] The meaning of the fields is described on the pages indicated in this column..

	Data structure kc_user_dyn2_str	Page[1]
	char us_name[8];	578
	union kc_pw password;	579
	char password_type;	580
	char password_dark;	580
B	char locale_lang_id[2];	581
B	char locale_terr_id[2];	581
B	char locale_ccsname[8];	581
	char protect_pw_lth;	581
	char protect_pw_compl;	582
	char protect_pw_time[3];	582
	char protect_pw_time_left[3];	585
	char protect_pw_min_time[3];	587
	char seccounter[5];	585
	char exclusively_signed;	589
	union kc_sign_date excl_sign_time_date;	589
	char node_last_excl_signon[4]	589
	char ptc;	590
	char bound_ptc;	590
	char bound_service;	590

[1] The meaning of the fields is described on the pages indicated in this column..

The fields in the data structures have the following meanings:

us_name
: Contains the name of the UTM user ID. The user specifies the user ID when signing on, and a UPIC client specifies the user ID when establishing a conversation with the application. *us_name* can be up to 8 characters long.

kset
: Contains the name of the key set assigned to the user ID. The key set determines the access privileges of the user within the application. The user can only call a service if both the key set of the user ID and the key set of the LTERM partner (by means of which the user connects to the application) contain a key or access code that corresponds to the lock code or access list of the requested service.

  The name of a key set can be up to 8 characters long.

  You can define a different key set in *kset* or remove the current key set by filling *kset* with blanks.

state
: Specifies if the user ID is currently permitted to sign on or connect, or if it is disabled.

  ´Y´ The user ID is allowed.

  ´N´ The user ID is currently disabled; no user or client may sign on to or establish a connection to the application with this user ID.

  The user ID can be disabled or permitted to sign on or connect again while the program is running. Disabling takes effect at the next sign-on attempt.

B card_position, card_string_lth, card_string_type, card_string
: You can determine if access to the application requires a magnetic strip card for this user ID using these fields. The fields specify which subfield of the identification information on the magnetic stripe will be checked and what information must be stored in this subfield.

B Specifying *card_xx* excludes the possibility of specifying *principal* and *certificate*.

B *card_position*
: Specifies the number of the byte at which the identification information to be checked begins; for example *card_position* = ´4´ means that the 4th byte of identification information corresponds to the 1st character of the section to be checked.

B *card_string_lth*
: Specifies how long the section of identification information to be checked is. The length is specified in bytes.

B *card_string_type*
B Specifies if the identification information to be checked is to be interpreted as a
B hexadecimal number or as a character string.

B ´X´ The identification information is a hexadecimal string.

B ´C´ The identification information is a string of printable, alphanumeric
B characters.

B ´N´ The user ID was configured without a magnetic strip card. In this case,
B *card_string_lth* and *card_position* contain ´0´ and blanks are returned in
B *card_string*.

B *card_string*
B The string that must be contained in the section to be checked on the magnetic
B stripe card in order for the user with this user ID to successfully sign on to the appli-
B cation.
B UTM returns the string in a union of type *kc_string*.

B
union kc_string
`char x[200];`
`char c[100];`

B If the identification information is a hexadecimal string (*card_string_type*=´X´), then
B each half byte is represented by one character.
B If *card_string_type*=´C´, then the contents of *card_string* are irrelevant after the length
B specified in *card_string_lth*.
B If *card_string_type*=´X´, then the contents of *card_string* are irrelevant after the length
B specified by 2 * *card_string_lth*.

password

password always contains blanks, even if a password is defined for the user ID, when information is queried with KC_GET_OBJECT.

The *password* field is only relevant for KC_MODIFY_OBJECT and KC_CREATE_OBJECT. You can then pass the new password for the user ID in *password* to UTM (see page 224 and 359).

B Specifying *password* excludes the possibility of specifying *certificate* and *principal*.

password_type

Specifies in a KC_GET_OBJECT call if a password was generated for the user ID.

´Y´ A password was generated for the user ID.

´N´ No password was generated for the user ID.

When changing a password with KC_MODIFY_OBJECT or when adding a new user ID, you specify the code used for the password in *password_type*.

´C´ The password is specified as a character string.

B ´X´ The password is specified as a hexadecimal number.

´N´ No password is specified.

password_dark

Specifies if the password must be hidden when entered at the terminal:

´Y UTM places the user in an intermediate dialog after signing on (KDCSIGN) in which the password is entered in a darkened field.

´N´ The user has to pass the password to UTM with the user ID when signing on (KDCSIGN). The password is not hidden when the user enters it.

X *For Unix and Linux systems only*:
X The entry specified in *password_dark* is ignored. The password is always
X non-displaying (“dark”). Whether or not the password has to be entered in
X a non-displaying field at sign-on via dialog terminal processes depends on
X the generation of the application. If the application is generated with
X formatting, the password must be entered in a non-displaying field.

B card_id
B Card identifier of the chip card.

B The user must identify with a chipcard on sign-on.

B In the case of operation code KC_GET_OBJECT, blanks are returned if the user
B has been generated without a chip card.

B format_attr, format_name
B These describe the user-specific start format. This start format is automatically
B output to the terminal after every successful sign-on if there are no open services
B for this user ID. If the user is still in a service after the access privileges have been
B successfully checked, then the start format does not appear, and the last dialog
B screen will be output instead (automatic restart).

format_attr
Contains the format code:

´A´ (format attribute ATTR)
The start format is a format with user attributes. The properties of the format fields can be changed by the KDCS program unit. The format name at the KDCS program interface is +*format_name*.

´N´ (format attribute NOATTR)
The start format is a format without user attributes. Neither the field nor the format properties can be changed by the KDCS program units. The format name at the KDCS program interface is **format_name*.

´E´ (format attribute EXTEND)
The start format is a format with expanded user attributes. The properties of the format fields as well as global format properties can be changed by the KDCS program unit. The format name at the KDCS program interface is #*format_name*.

format_name
Contains the name of the start format. The name can be up to 7 characters long and contains only alphanumeric characters.

locale_lang_id, locale_terr_id, locale_ccsname
These contain the three components of the locale assigned to the user ID. The locale defines the language environment of the users/clients that connect to the application via this user ID (see also the openUTM manual “Generating Applications”).

locale_lang_id
Contains the up to two characters long language code.

locale_terr_id
Contains an up to two characters long territory code.

locale_ccsname
(**c**oded **c**haracter **s**et **name**)
Contains the up to 8 characters long name of an extended character set (CCS name; see also the XHCS User Guide).

protect_pw_lth
Specifies the minimum number of characters the password for the user ID must have in order for it to be accepted by UTM (minimum length of the password). The administrator can only delete the user’s password if ´0´ is returned in *protect_pw_lth*.

protect_pw_compl
: Specifies the complexity level the password for the user ID must have.

  ´0´ (NONE)
  Any string can be specified as the password.

  ´1´ (MIN)
  A maximum of two characters in a row may be exactly the same in the password.

  ´2´ (MEDIUM)
  A maximum of two characters in a row may be exactly the same in the password. The password must contain at least one letter and one number.

  ´3´ (MAX)
  A maximum of two characters in a row may be exactly the same in the password. The password must contain at least one letter, one number and one special character. Special characters are all characters not in a-z, A-Z, and 0-9. The space character is also a special character.

protect_pw_time
: Specifies the maximum number of days the password is valid (duration of validity).

  The validity of the password runs out at the end of the last day of the duration of validity. If, for example, a password is generated with a validity of one day, then the validity will run out at 24:00 hrs. on the following day.

  Shortly before the validity runs out, UTM requests the user to change the password with the K121 UTM message.

  If the validity runs out, the following applies:
  If the grace sign-on is generated (*kc_signon_str.grace*=´Y´) the user can change the password when next signing on.
  If the grace sign-on is not generated, UTM will reject an attempt to sign on and issues message K120. The administrator must then change the password.

  protect_pw_time = ´0´ means that the password is valid indefinitely.

restart
: Specifies whether UTM executes an automatic restart for this user ID.

  ´Y´ UTM executes an automatic restart for users who sign on using this user ID.

  UPIC client that are signed on to UTM under this user ID can initiate the restart of an open service when a new connection is established by sending the KDCDISP command.

´N´ UTM does not execute an automatic restart for users who sign on using this user ID.

If the application is generated with SIGNON MULTI-SIGNON=YES, several users/clients can be signed on under this user ID at the same time. Only one of these users may be signed on at the terminal. Any number of UPIC clients, TS applications and OSI-TP partners can be signed on at the same time under this user ID, however.

permit Specifies which privileges the user ID has within the local application.

´A´ (ADMIN)
The user ID has administration privileges, i.e. all administration functions in the local application may be executed by this user ID.

´N´ (NONE)
The user ID does not have administration privileges.

If the local application is a UTM application on a BS2000 system, UTM SAT administration functions are also not permitted to be executed under this user ID.

´B´ (BOTH)
Administration functions and UTM SAT administration functions may be executed in the local application under this user ID.

´S´ (SAT)
The user ID has UTM SAT administration privileges. Preselection functions may be executed under this user ID, i.e. the SAT logging can be enabled or disabled for certain events.

satsel Specifies which events SAT will log for this user ID. One requirement for logging is that SAT logging is enabled for the application (*kc_max_par_str.sat*=´Y´). See also the openUTM manual “Generating Applications” and openUTM manual “Using openUTM Applications on BS2000 Systems” for more information on SAT logging.

´B´ (BOTH)
Both successful **and** unsuccessful events are logged.

´S´ (SUCCESS)
Only successful events are logged.

´F´ (FAIL)
Only unsuccessful events are logged.

´N´ (NONE)
No user-defined type of SAT logging is defined.

user_type

Specifies the type of client for which the LTERM partner is created for user IDs that are assigned to an LTERM partner.

´A´ (APPLI)
The user ID is assigned to the LTERM partner of a TS application of the type APPLI (PTERM with PTYPE=APPLI).

´S´ The user ID is assigned to the LTERM partner of a socket application (PTERM with PTYPE= SOCKET).

´U´ (UPIC)
The user ID is assigned to the LTERM partner of a UPIC clients (PTERM with PTYPE=UPIC-R or UPIC-L).

For all other user IDs a blank will be returned in *user_type*.

lterm_curr

The following cases must be distinguished:

The application is generated with SIGNON MULTI-SIGNON=NO
(i.e. multiple sign-ons are not allowed):
lterm_curr contains the LTERM partner or the OSI-LPAP partner through which a user with this user ID is signed on.
Exception: *lterm_curr* contains blanks if the sign-on is to start an asynchronous service via OSI TP.

The application is generated with SIGNON MULTI-SIGNON=YES
(multiple sign-ons are possible):

– If a user with this user ID is connected to the application via a terminal, then *lterm_curr* contains the name of the LTERM partner assigned to the terminal.

– If the user ID is generated with *restart*=´Y´, then *lterm_curr* contains the name of the LTERM or OSI-LPAP partner through which a client with this user ID is connected.
Exception: signing on is handled via OSI TP and the functional unit “commit” was selected, or signing on is handled via OSI TP to start an asynchronous service. In this case *lterm_curr* contains blanks.

In all other cases *lterm_curr* contains blanks.

connect_mode

Specifies whether a user or a client with this user ID is currently connected through the LTERM or OSI-LPAP partner in *lterm_curr* (´Y´) or not (´N´).

in_service
: Specifies whether a service is currently running under this user ID through the LTERM or OSI-LPAP partner in *lterm_curr*.

  ´Y´ A service is open which has reached at least one consistency point.

  ´N´ Currently no service is running which has reached at least one consistency point.

number_tacs
: Contains the number of program units executed under this user ID. In UTM-S applications, the value of *number_tacs* is reset to 0 in each regeneration with KDCDEF or in each update generation with KDCDEF/KDCUPD. In UTM-F applications, *number_tacs* is reset to 0 each time the application is started.

cputime_sec
: Contains the number of CPU used for processing jobs for this user ID since the last connection establishment. However, the value returned in *cputime_sec* does not contain the CPU time used for database calls.

seccounter
: Contains the number of security breaches for this user ID (e.g. incorrect password, illegal transaction code) since the application was last started.

deleted
: Specifies whether the user ID was deleted from the configuration or not.

  ´Y´ The user ID was deleted with a delay (KC_DELAY). However, the name is still disabled, i.e. you cannot create a new user ID with this name.

  ´N´ The user ID was not deleted.

protect_pw_time_left
: For opcode KC_GET_OBJECT:
  Specifies for how much longer the current password is valid. *protect_pw_time_left* specifies the period in days.

  The following values are also possible:

  ´ ´ (Blanks)
  No password was generated for this user ID or the password was deleted.

  ´000´ The password expires on the current day.

  ´-1´ A password with an indefinite term of validity was assigned to this user ID (*protect_pw_time*=´0´).

  ´-2´ The term of validity of the password has aready expired.

For opcode KC_MODIFY_OBJECT:
Only relevant in applications generated with SIGNON GRACE=YES and for user IDs for which a restricted password validity period has been generated.
In *protect_pw_time_left*, you specify whether the generated period of validity is to apply to the new password. Any specification in this field is ignored unless there is also a specification for *password* and *password_type*.
If you specify *protect_pw_time*=-1'(left or right-aligned) then the generated period of validity (starting from the time of the modification) applies for the new password.
If you do not specify anything then the new password is immediately invalid due to the expiry of the period of validity. The user must change the password the next time he or she signs on.
Any value other than '-1' is rejected.

sign_time_date

Specifies when a user or client last signed on with UTM using this user ID.

UTM returns the date and time at which a user last signed on in the field *cstring* of a union to the type *kc_sign _date*.

union kc_sign_date
`char cstring[14];`
`struct cstr_str cstring_struct;`

where

struct cstr_str
`char year[4];`
`char month [2];`
`char day[2];`
`char hour[2];`
`char minute[2];`
`char sec[2];`

The data is output in the format ´YYYYMMDDhhmmss´, *YYYY* being the year, *MM* the month, *DD* the day, *hh* the hour, *mm* the minute and *ss* the second.

If no user or client has as yet signed on with the application using this user ID, UTM returns ´00000000000000´.

asyn_services

Contains the number of asynchronous jobs currently running for this user ID.

clients_signed
: Contains the number of communication partners currently signed on at the application under this user ID.

  The value may be temporarily greater than 1 even in applications generated with SIGNON MULTI-SIGNON=NO if an OSI TP communication partner is currently signed on under this user ID for the generation of an asynchronous job.

protect_pw_min_time
: Specifies the minimum term of validity of the password in days.
  After the password has been changed, the user cannot change it again before this minimum period has expired.

  The user can always change the password after it has been previously changed by the administrator or after a regeneration, regardless of whether or not the minimum term of validity has expired.

qlev (**q**ueue **l**evel)
: Indicates the maximum number of messages that can be stored in the queue of the user. If the threshold value is exceeded, the response of openUTM depends on the value in the *q_mode* field.

  UTM ignores the messages created for the queue until the end of the transaction. The number of messages for a message queue specified in *qlev* can therefore be exceeded if several messages are created for the same queue in a single transaction.

  If *qlev*=0 is specified, no messages can be stored in the queue. If *qlev*=32767 is specified, there is no limit on the queue length.

out_queue
: Indicates the number of messages in the user's message queue.

  For more detailed information, refer to the openUTM manual "Generating Applications".

  If the number of messages is greater than 99999, then the number is not displayed in full. You should therefore use the field *out_queue_ex* or the field *out_queue* from the data structure *kc_user_dyn1* since larger numbers can be entered in full here.

q_read_acl
: Indicates the rights (name of a key set) required by another user in order to read and delete messages from the user queue.
  Another user can only have read access to this queue if the key set of the user's user ID and the key set of the LTERM partner by means of which the user is signed on each have at least one key code that is also contained in the displayed key set.

  If *q_read_acl* does not contain a value, all users can read and delete messages from this queue.

q_write_acl

Indicates the rights (name of a key set) that another user requires in order to write messages to this user queue.
Another user can only have write access to this queue if the key set of the user's user ID and the key set of the LTERM partner by means of which the user is signed on each have at least one key code that is also contained in the displayed key set.

If *q_write_acl* does not contain a value, all users can write messages to this queue.

q_mode (**q**ueue **m**ode)

Indicates how UTM responds if the maximum number of as yet unexecuted jobs is reached in the queue of the user (see *qlev*). Possible value are:

´S´ UTM rejects any further messages.

´W´ UTM accepts any further messages. However, when a new message is written to the queue, the oldest message in the queue is deleted.

B certificate

B Specifies the certificate number with which the user must identify himself or herself
B when signing on to the UTM application.

B '*NONE' No certificate is used for this USER for the purpose of identification.

B You will find more information in the openUTM manual "Generating Applications".

B Specifying *certificate* excludes the possibility of specifying *principal*, *card_xx* and
B *password*.

B cert_auth

B Specifies the number of a certification point.

B You will find more information in the openUTM manual "Generating Applications".

bcam_trace

Specifies whether the BCAM trace is explicitly enabled for this USER.

'Y' The BCAM trace is explicitly enabled for this USER.

'N' The BCAM trace is not explicitly enabled for this USER.

It only makes sense to evaluate the field using KC_GET_OBJECT if the BCAM trace is enabled for individual USERs. If the BCAM trace is generally enabled (see *kc_diag_and_account_par_str*) *bcam_trace*='N' is returned here for this user.

The BCAM trace can be explicitly enabled or disabled by calling KC_MODIFY_OBJECT. The BCAM trace can then only be enabled for individual USERs

– if it is disabled for all USERs (see *kc_diag_and_account_par_str*) or
– if it has only been enabled for individual USERs up to now.

B principal
B The user is authenticated using Kerberos. It is only possible to authenticate users
B using Kerberos if the user signs in directly (not via OMNIS) at a terminal that
B supports Kerberos.

B Specifying *principal* excludes the possibility of specifying *certificate*, *card_xx* and
B *password*.

B If a query is issued with KC_GET_OBJECT, the principal is displayed here if the
B user has been generated with Kerberos authentication.

B When calling KC_CREATE_OBJECT, you enter an alphanumeric string of the
B following form here:

B `windowsaccount@NT-DNS-REALM-NAME'`

B windowsaccount
B Domain account of the user

B NT-DNS-REALM-NAME
B DNS name of the Active Directory domain. This name is a fixed value for
B every Active Directory domain and was assigned when the Kerberos key
B was set up.

node_last_excl_signon
This field is only relevant for UTM cluster applications.
Number (index) of the node application that a user/client with this user ID was most recently exclusively signed on to.

exclusively_signed
This field is only relevant for UTM cluster applications.
exclusively_signed specifies whether a user/client is currently signed on exclusively with this user ID.

´Y´ The user/client is currently signed on exclusively.

´N No user/client is signed on exclusively with the user ID.

excl_sign_time_date
This field is only relevant for UTM cluster applications.
Date and time that this user most recently signed on exclusively.

UTM returns the date and time of the last sign-on in a union of type *kc_sign_date*.

union kc_sign_date
`char cstring[14];`
`struct cstr_str cstring_struct;`

where

struct cstr_str
`char year[4];`
`char month [2];`
`char day[2];`
`char hour[2];`
`char minute[2];`
`char sec[2];`

The output has the form ´YYYYMMDDhhmmss´. Where *YYYY* is the year, *MM* the month, *DD* the day, *hh* the hour, *mm* the minute and *ss* the second.

If no user or client has yet signed on exclusively with the user ID, openUTM returns ´00000000000000´.

out_queue_ex
: see *out_queue* on page 587.

ptc
: The user has an open service with a transaction in the PTC state

bound_ptc
: The user has a node-bound service with a transaction in the PTC state (relevant only for UTM cluster applications).

bound_service
: The user had a node-bound service on the last sign-off (relevant only for UTM cluster applications).

cputime_msec
: Indicates the number of CPU milliseconds used since the last establishment of a connection for the processing of jobs for this user ID. However, the value returned in *cputime_msec* does not include the CPU time used for database calls.

11.3.2 Data structures used to describe the application parameters

All data structures that are provided for passing application parameters are described in the following section. Every single parameter type is provided its own data structure in the *kcadminc.h* header file. The name of the corresponding data structure is created from the name of the parameter type and the suffix "*_str*". The descriptions are listed in alphabetically ascending order according to the names of the data structures.

kc_cluster_curr_par_str - Statistics values of a UTM cluster application

The data structure *kc_cluster_curr_par_str* is defined for the object type KC_CLUSTER_CURR_PAR. In the case of KC_GET_OBJECT, UTM returns information on the utilization of the cluster page pool in *kc_cluster_curr_par_str*.

KC_MODIFY_OBJECT can be used to reset the counters to 0.

mod[1]	Data structure kc_cluster_curr_par_str
x(GID)	`char max_cpgpool_size[3];`
–	`char curr_cpgpool_size[3];`
x(GID)	`char avg_cpgpool_size[3];`
–	`char node_reserved_cpgpool_pages[10];`

[1] The content of the field can be modified using KC_MODIFY_OBJECT; see page 362f

The fields in the data structure have the following meanings:

max_cpgpool_size
: Specifies the maximum cluster page pool utilization in %.

  The value continues to apply after the entire UTM application run. It is reset when the size of the cluster page pool is increased and when the UTM cluster files are generated using KDCDEF.

  KC_MODIFY_OBJECT:
  Resets the value to 0. This also implicitly resets the value of *avg_cpgpool_size* to 0.

curr_cpgpool_size
: Specifies the current cluster page pool utilization in %.

avg_cpgpool_size
: Specifies the average cluster page pool utilization in %.

  The value continues to apply after the entire UTM application run. It is reset when the size of the cluster page pool is increased and when the UTM cluster files are generated using KDCDEF.

  KC_MODIFY_OBJECT:
  Resets the value to 0. This also implicitly resets the value of *max_cpgpool_size* to 0.

node_reserved_cpgpool_pages
: Specifies the number of reserved pages for the current local node.

kc_cluster_par_str - Global properties of a UTM cluster application

The data structure *kc_cluster_par_str* is defined for the parameter type KC_CLUSTER_PAR. In the case of KC_GET_OBJECT, UTM uses *kc_cluster_par_str* to return the current settings for the properties of a UTM cluster application together with current data (e.g. generation time, start time, number of active and generated node applications).
You can use KC_MODIFY_OBJECT to modify the following:

- Parameters which control the verification of the individual node applications
- Parameters which control node application access to the cluster configuration file and the cluster administration journal.

mod[1]	data structure kc_cluster_par_str
–	`struct kc_cluster_filebase cluster_filebase;`
–	`struct kc_admi_date_time_model gen_time;`
–	`char os_type[24];`
–	`char bit_mode[8];`
–	`char bcamappl[8];`
–	`char port_nbr[8];`
x(GID)	`char check_alive_timer_sec[8];`
x(GID)	`char communication_retry[8];`
x(GID)	`char communication_reply_timer_sec[8];`
x(GID)	`char restart_timer_sec[8]:`
x(GID)	`char file_lock_timer_sec[8];`
x(GID)	`char file_lock_retry[8];`
–	`char max_nbr_nodes[4];`
–	`char curr_nbr_nodes[4];`
–	`char nbr_active_nodes[4];`
–	`char emergency_cmd [200];`
–	`char failure_cmd [200];`
–	`struct kc_admi_date_time_model last_kdcdef_time;`
B –	`char xcs_name[8];`
–	`struct kc_admi_date_time_model cluster_start_time;`
–	`char abort_bound_service;`
x(GID)	`char deadlock_prevention;`
–	`char listener_id[5];`

mod[1]	data structure kc_cluster_par_str
–	`char cpgpool[10];`
–	`char cpgpool_warnlevel[2];`
–	`char cpgpool_fs[2];`

[1] Field content can be modified with KC_MODIFY_OBJECT , see page 311f

The fields in the data structure *kc_cluster_par_str* correspond to the configuration information in the KDCDEF control statement CLUSTER, see openUTM manual “Generating Applications”.

The fields in the data structure have the following meanings:

cluster_filebase
: Name prefix or directory (base name) of the cluster configuration file and other global administration files of the UTM cluster application, e.g. the administration journal.

  The name is passed in the element *cluster_filebase* of type *kc_cluster_filebase*:

struct kc_cluster_filebase
`char length[2];`
`char fb_name[54];`

  fb_name contains the base name and *length* the length of the base name.

gen_time
: Time at which the cluster configuration file was generated. The date and time are returned in the element *gen_time* of type *kc_admi_date_time_model*.

struct kc_admi_date_time_model
`struct kc_admi_date_model admi_date;`
`struct kc_admi_time_model admi_time`

  where

struct kc_admi_date_model
`char admi_day [2];`
`char admi_month [2];`
`char admi_year_4 [4];`
`char admi_julian_day [3];`
`char admi_daylight_saving_time`

and

struct kc_admi_time_model
`char admi_hours [2];`
`char admi_minutes [2];`
`char admi_seconds [2]`

os_type
: System platform of the computer, e.g. SolarisSparc"

bit_mode
: Mode in which the operating system is running. The following values are returned:

  '32 Bit' for 32-bit mode.
  '64 Bit' for 64-bit mode.

bcamappl
: Name of the transport system endpoint (BCAMAPPL name) that is used for communication within the cluster.

port_nbr
: Number of the listener port used for communication within the cluster.

check_alive_timer_sec
: In a UTM cluster application, every node application is monitored by another node application (circular monitoring), i.e. each node application monitors the availability of another node application and is itself monitored by a node application. To do this, the monitoring node application sends messages to the monitored node application at defined intervals (*check_alive_timer_sec*). If the monitored application is available, it acknowledges the message.
  check_alive_timer_sec specifies the interval in seconds at which monitoring messages are sent to the monitored node application.
  This timer is also used for periodic access to the cluster configuration file and the cluster administration journal.

  KC_MODIFY_OBJECT:
  You can modify the monitoring interval.

  Minimum value: '30'
  Maximum value: '3600'

communication_retry

Specifies how often a node application repeats an attempt to send a monitoring message if the monitored node application does not respond within the time defined in *communication_reply_timer_sec*.
If the monitored node application does not respond to any of the retries in the defined time, then it is assumed to have failed and the command sequence defined in *failure_cmd* is executed (e.g. a restart).

KC_MODIFY_OBJECT:
You can modify the value of *communication_retry*.

Minimum value: ‘0'
Maximum value: ‘10'

communication_reply_timer_sec

Maximum time in seconds that a node application waits for a response after sending a monitoring message.
If no response is received within this period then the monitored node application is assumed to have failed (abnormal end of application) and the command sequence defined in *failure_cmd* is executed (e.g. a restart).

If a value greater than zero is set for *communication_retry*, then the target node application is only assumed to have failed if, additionally, no response to the monitoring message is received after the final retry.

KC_MODIFY_OBJECT:
You can modify the settings for *communication_reply_timer_sec*.

Minimum value: ’1’
Maximum value: ’60’

restart_timer_sec

Maximum time in seconds that a node application requires for a warm start after a failure (abnormal program termination).

The monitoring node application waits for the time specified here after calling the command sequence specified under *failure_cmd* before sending another monitoring message to this node application. If the monitoring node application does not receive a response to this message, it is assumed that the failed node application can no longer be restarted as a result of a persistent problem. The command sequence specified in *emergency_cmd* is called for the failed node application.

KC_MODIFY_OBJECT:
You can modify the value of *restart_timer_sec*.

Minimum value: ‘0’, i.e. no time monitoring of restart.
Maximum value: ’3600’

file_lock_timer_sec
: Maximum time in seconds that a node application waits for a lock to be assigned for accessing the cluster configuration file of the cluster administration journal.
file_lock_retry specifies how often a node application repeats the request for a lock on the cluster configuration file or the cluster administration journal if the lock was not assigned in the time specified in *file_lock_timer_sec*.

KC_MODIFY_OBJECT:
Sets a new value for file_lock_timer_sec.

Minimum value: ‘10'
Maximum value: ‘60'

file_lock_retry
: Specifies how often a node application repeats the request for a lock on the cluster configuration file or the cluster administration journal if the lock was not assigned in the time specified in *file_lock_timer_sec*.

KC_MODIFY_OBJECT:
You can modify the value of *file_lock_retry*:

Minimum value: ‘1'
Maximum value: ‘10'

max_nbr_nodes
: Maximum possible number of node applications that can be generated in a UTM cluster application.

B
B
: In an XCS cluster of BS2000 systems, a maximum of 16 of the 32 node applications that can be generated can run at any one time.

curr_nbr_nodes
: Number of node applications actually generated for this UTM cluster application (corresponds to the number of CLUSTER-NODE statements in the KDCDEF generation of the UTM cluster application).

nbr_active_nodes
: Number of node applications currently active (started) in the UTM cluster application.

emergency_cmd
: Contains a command to be executed together with its arguments.
This command is called by UTM if a failed node application cannot be restarted and a value greater than zero has been set for *restart_timer_sec*. I.e., the actions specified in *failure_cmd* have not resulted in the failed node application being restarted (in time).

failure_cmd
: Contains a command to be executed together with its arguments. This command is called by UTM if a node application terminates abnormally or if failure of a node application is detected. The command in *failure_cmd* can, for example, be used to initiate the restart of a failed node application or to send an e-mail to the system administrator.

last_kdcdef_time
: Time of the last generation of a KDCFILE which has been used to start at least one node application.

  The date and time are returned in the element *last_kdcdef_time* of type *kc_admi_date_time_model* (see *gen_time* page 594).

B B xcs_name
: Name of the XCS computer cluster on which the UTM cluster application is running.

cluster_start_time
: Time at which the first node application in the UTM cluster application was started.

  The date and time of the start are returned in the element *cluster_start_time* of type *kc_admi_date_time_model* (see *gen_time* page 594).

abort_bound_service
: ´N´ If when a user signs on, there is an open service for this user that is bound to another node application, then the user can only sign on at the node application to which the open service is bound. Sign-on attempts at any other node application are rejected.

  ´Y´ If when a user signs on at a node application, there is an open service for this user that is bound to another node application that has been terminated, then the user is able to sign on provided that no transaction of the open service has the state PTC. No service restart is performed

  The open service is terminated abnormally the next time the node application to which it is bound is started.

deadlock_prevention
: Specifies whether or not UTM is to perform additional checks of the GSSB, TLS and ULS data areas in order to prevent deadlocks.

  ´N´ UTM does not perform any additional checks of the GSSB, TLS and ULS data areas in order to prevent deadlocks If a deadlock occurs in one of these data areas then this is resolved by means of a timeout.

  ´Y´ UTM performs additional checks of the GSSB, TLS and ULS data areas in order to prevent deadlocks.

  In productive operation, it is advisable to set this parameter to ´Y´ only if timeouts occur frequently when accessing these data areas.

X/W
X/W
X/W
listener_id
: This parameter is used to select a network process for internal cluster communication.

cpgpool
: Size of the cluster page pool in 4K pages.

cpgpool_warnlevel
: Percentage value specifying the cluster page pool utilization level at which a warning (message K041) is output.

cpgpool_fs
: Number of files over which the user data is distributed in the cluster page pool.

kc_curr_par_str - Current values of the application parameters

The data structure *kc_curr_par_str* is defined for the parameter type KC_CURR_PAR. In the case of KC_GET_OBJECT, UTM returns the current values of the parameter settings, data pertaining to the application run and statistical information on the load of the application in *kc_curr_par_str* (see also KDCINF, "type=STATISTICS" on page 726).

You can reset some of the counters used by UTM to generate statistical information with the aid of KC_MODIFY_OBJECT if you need to (see also *max_statistics_msg* page 638).

If MAX STATISTICS-MSG=NONE the counters in a UTM-S application are only reset the first time the application is started and in UTM-F applications they are reset each time the application is started.

If MAX STATISTICS-MSG=FULL-HOUR then the counters in a UTM-S application are reset every full hour. As a result, the values displayed in the initial period following a full hour may be too low.

mod[1]	Data structure kc_curr_par_str
–	`char appliname[8];`
–	`char utm_version[8];`
–	`char applimode;`
–	`char start_date_year[4];`
–	`char start_date_month[2];`
–	`char start_date_day[2];`
–	`char start_time_hour[2];`
–	`char start_time_min[2];`
–	`char start_time_sec[2];`
–	`char curr_date_year[4];`
–	`char curr_date_month[2];`
–	`char curr_date_day[2];`
–	`char curr_time_hour[2];`
–	`char curr_time_min[2];`
–	`char curr_time_sec[2];`
x(GIR)	`char term_input_msgs[10];`
x(GIR)	`char term_output_msgs[10];`
–	`char curr_max_asyntasks[3];`
–	`char curr_max_tasks_in_pgwt[3];`
–	`char curr_tasks[3];`
–	`char curr_asyntasks[3];`

mod[1]	Data structure kc_curr_par_str
–	char curr_tasks_in_pgwt[3];
–	char tasks_waiting_in_pgwt[3];
–	char connected_users[10];
–	char *rvices[10];
–	char open_asyn_services[10];
–	char dial_ta_per_100sec[10];
–	char asyn_ta_per_100sec[10];
–	char dial_step_per_100sec[10];
x(GIR)	char max_dial_ta_per_100sec[10];
x(GIR)	char max_asyn_ta_per_100sec[10];
x(GIR)	char max_dial_step_per_100sec[10];
x(GIR)	char max_pool_size[3];
–	char curr_pool_size[3];
x(GIR)	char avg_pool_size[3];
x(GIR)	char cache_hit_rate[3];
x(GIR)	char cache_wait_buffer[3];
–	char unproc_atacs[10];
–	char unproc_prints[10];
–	char wait_dputs[10];
x(GIR)	char abterm_services[10];
–	char wait_resources[4];
x(GIR)	char deadlocks[10];
x(GIR)	char periodic_writes[10];
x(GIR)	char pages_pwrite[10];
x(GIR)	char logfile_writes[10];
–	char curr_jr[3];
x(GIR)	char maximum_jr[3];
–	char program_fgg[4];
–	char uslog_fgg[4];
x(GIR)	char max_mpgpool_size[3]; [2]
–	char curr_mpgpool_size[3]; [2]
x(GIR)	char avg_mpgpool_size[3]; [2]

mod[1]	Data structure kc_curr_par_str
x(GIR)	`char max_load[3];`
–	`char curr_load[3];`
x(GIR)	`char max_wait_resources[4];`
–	`char wait_system_resources[4];`
x(GIR)	`char max_wait_system_resources[4];`
x(GIR)	`char nr_cache_rqs[10];`
x(GIR)	`char nr_cache_searches[10];`
–	`char nr_res_rqs[10];`
x(GIR)	`char nr_res_rqs_for_max[10];`
–	`char nr_sys_res_rqs[10];`
x(GIR)	`char nr_sys_res_rqs_for_max[10];`
–	`char curr_system_tasks[3];`
x(GID)	`char data_compression;`
x(GIR)	`char avg_saved_pgs_by_compr[3];`

[1] The field contents can be modified with KC_MODIFY_OBJECT; see page 365f

[2] Internal UTM field; the contents of the field are irrelevant and will not be described in the following.

The fields in the data structure have the following meanings:

appliname
: Name of the UTM application set in the KDCDEF generation in MAX APPLINAME.

  appliname is the name of the application that must be specified when establishing a connection from the terminal.

utm_version
: The UTM version used including the update information, for example V06.4A00.

applimode
: Specifies if the UTM application is a UTM-S or UTM-F application.

  ´S´ The application is generated as a UTM-S application (secure).

  ´F´ The application is generated as a UTM-F application (fast).

start_date_year, start_date_month, start_date_day
: UTM-S application: date of the last cold start of the application
  UTM-F application: date of the last start of the application

start_time_hour, start_time_min, start_time_sec
: UTM-S application: time of the last cold start of the application
  UTM-F application: time of the last start of the application

curr_date_year, curr_date_month, curr_date_day
: The current date.

curr_time_hour, curr_time_min, curr_time_sec
: The current time.

term_input_msgs
: Total number of messages that the application has received from clients or partner applications since the last time the *term_input_msgs* counter was reset.
UTM automatically resets the counter to 0 each time the application is started and on each full hour, if MAX STATISTICS-MSG=FULL-HOUR (default value) was set during KDCDEF generation.
You can set *term_input_msgs* to 0.

term_output_msgs
: Total number of messages that the application sent to clients, printers or partner applications since the last time the *term_output_msgs* counter was reset.

  UTM automatically resets the counter to 0 each time the application is started and on each full hour, if MAX STATISTICS-MSG=FULL-HOUR (default value) was set during KDCDEF generation.

  You can set *term_output_msgs* to 0.

curr_max_asyntasks
: Current setting for the maximum number of processes that may be used for asynchronous processing. *curr_max_asyntasks* is dynamically adjusted by UTM if the total number of processes of the application or the maximum number of processes for asynchronous processing (*kc_tasks_par_str.mod_max_asyntasks* on page 654) is changed by the administration.

curr_max_tasks_in_pgwt
: Current setting for the maximum number of processes that may simultaneously process jobs from TAC classes whose transaction codes are allowed to use blocking calls such as, for example, the KDCS call PGWT (Program Wait).
curr_max_tasks_in_pgwt is dynamically adjusted by UTM if the total number of processes of the application or the number of processes *kc_tasks_par.mod_max_tasks_in_pgwt* (see page 654) is changed.

curr_tasks
: Contains the number of processes of the application currently running.

curr_asyntasks
: Contains the number of processes currently processing asynchronous jobs.

curr_tasks_in_pgwt
: Contains the number of processes currently processing jobs whose transaction codes are allowed to use blocking function calls (for example PGWT).

tasks_waiting_in_pgwt
: The current number of processes in the wait state due to blocking function calls (for example the KDCS call PGWT).

connected_users
: The number of users currently connected to the application.

open_dial_services
: The number of dialog services currently open.
In a UTM cluster application, an open dialog service that is valid globally in the cluster is only counted if the user is signed on.

open_asyn_services
: The number of asynchronous services currently open.

dial_ta_per_100sec
: The current number of dialog transactions executed in the last closed 100 second interval.

asyn_ta_per_100sec
: The current number of asynchronous transactions executed in the last closed 100 second interval.

dial_step_per_100sec
: The current number of dialog steps executed in the last closed 100 second interval.

max_dial_ta_per_100sec
: The maximum number of dialog transactions that were executed within a 100 second interval. The value is specified for the current application run.
It can be reset with KC_MODIFY_OBJECT (see page 365).

max_asyn_ta_per_100sec
: The maximum number of asynchronous transactions that were executed within a 100 second interval. The value is specified for the current application run. It can be reset with KC_MODIFY_OBJECT (see page 365).

max_dial_step_per_100sec
: The maximum number of dialog steps that were executed within a 100 second interval. The value is specified for the current application run. It can be reset with KC_MODIFY_OBJECT (see page 365).

max_pool_size
: The maximum amount of the page pool in use in percent. For UTM-S applications the maximum page pool size since the most recent KDCDEF generation is returned, for UTM-F applications the size since the last application start is returned.
The value can be reset with KC_MODIFY_OBJECT (see page 365).

curr_pool_size
: The current amount of the page pool in use in percent.

avg_pool_size
: For UTM-S applications the maximum page pool size since the most recent KDCDEF generation is returned, for UTM-F applications the size since the last application start is returned. The value can be reset with KC_MODIFY_OBJECT (see page 365).

cache_hit_rate
: The hit rate when searching for a page in the cache. Specified in percent.The value refers to the current application run. It can be reset with KC_MODIFY_OBJECT (see page 365). If this value is reset then the values *cache_wait_buffer*, *nr_cache_rqs* and *nr_cache_searches* are also implicitly reset to 0.

cache_wait_buffer
: The percentage of queries from buffers in the cache that have resulted in a wait state. *cache_wait_buffer* gives you the amount of buffer queries since the counter was last reset.
UTM automatically resets the counter to 0 each time application is started and on each hour, if MAX STATISTICS-MSG=FULL-HOUR (default value) was generated in the KDCDEF generation.
You can reset the counter using KC_MODIFY_OBJECT (see page 365). If this value is reset then the values *cache_hit_rate*, *nr_cache_rqs* and *nr_cache_searches* are also implicitly reset to 0.

unproc_atacs
: The number of background jobs currently stored in UTM but not yet completely processed. This corresponds to the number of messages temporarily stored at the present time in all of the message queues of asynchronous services.

unproc_prints
: The number of messages temporarily stored at the present time in the message queues of all of the printers.

wait_dputs
: The number of time-driven jobs currently waiting (DPUTs).

abterm_services
: The number of abnormally terminated services since the value was last reset.You can reset *abterm_services* with KC_MODIFY_OBJECT.

wait_resources
: This value indicates the mean lock conflict rate for the GSSB, ULS and TLS memory areas during the last closed 100 second interval as an amount per thousand, i.e. the total number of wait situations on lock requests as a ratio of GSSB, ULS and TLS lock requests in the last closed 100 second interval multiplied by 1000.

A higher value in *wait_resources* can be caused by the following:

– processes with run times or wait times that are too long
– resources that have been locked for too long, for example, due to many PEND KP or PGWT calls in KDCS program units.

If a lock holder enters the status PEND KP then all "waiters" are informed and all further locks are rejected immediately. I.e. the value of *wait_resources* does not increase as a result.

deadlocks
The number of deadlocks of UTM resources that have been recognized and resolved since the value was last reset.

You can reset *deadlocks* using KC_MODIFY_OBJECT.

periodic_writes
The number of periodic writes since the last start of the application or since the value was last reset with KC_MODIFY_OBJECT. (periodic write = the saving of all relevant administration data of the UTM application.)

pages_pwrite
The number of UTM pages that are saved during a periodic write on the average. All periodic writes since the value was last reset are registered. You can reset the value using KC_MODIFY_OBJECT. UTM automatically resets *pages_pwrite* to zero each time the application is started.

logfile_writes
The number of request to write log entries to the user log file (USLOG) since the value was last reset
UTM automatically resets the counter to 0 each time application is started and on each hour, if MAX STATISTICS-MSG=FULL-HOUR (default value) was generated in the KDCDEF generation.

You can reset the counter using KC_MODIFY_OBJECT (see page 365).

curr_jr Only for distributed processing:
The current number of simultaneously addressed job-receiving services relative to the generated value MAXJR in percent.

(MAXJR = maximum number of remote job-receiving services that may be addressed simultaneously in the local application; see *kc_utmd_par_str* on page 661).

maximum_jr
: Only in the case of distributed processing:

  The current number of simultaneously addressed job-receiving services in the local application relative to the generated value MAXJR (see *kc_utmd_par_str* on page 661). Specified in percent.

  maximum_jr returns all requests to the remote job-receiving service since the value was last reset. You can reset *maximum_jr* to zero with KC_MODIFY_OBJECT.

program_fgg
: X/W On Unix, Linux and Windows systems: The number of file generations of the application program currently loaded.

  X/W

  B On BS2000 systems: 0

uslog_fgg
: The number of file generations of the user log file (USLOG) currently being written to.

max_load
: Indicates as a percentage the maximum load on the UTM application since the start of the application or the last reset was registered.

  The value in *max_load* can be reset to the value in *curr_load*.

curr_load
: Indicates as a percentage the current load on the UTM application registered during the last closed 100 second intervall.

max_wait_resources
: Maximum conflict rate for user data locks across the application run. The value is specified as an amount per thousand.

  You can reset this value with KC_MODIFY_OBJECT. If this value is reset then the values *max_wait_system_resources*, *nr_res_rqs_for_max* and *nr_sys_res_rqs_for_max* are also implicitly reset to 0.

wait_system_resources
: Average conflict rate in the last closed 100 second interval for the most heavily loaded system resource during this interval. The output can refer to different system resources in different intervals. The value is specified as an amount per thousand.

max_wait_system_resources
: Maximum conflict rate for system resource requests (system locks) across the application run. The value is specified as an amount per thousand.

  You can reset this value with KC_MODIFY_OBJECT. If this value is reset then the values *max_wait_resources nr_res_rqs_for_max* and *nr_sys_res_rqs_for_max* are also implicitly reset to 0.

nr_cache_rqs
: Number of buffer requests taken into account to calculate the value *cache_wait_buffer*.

  You can reset the value using KC_MODIFY_OBJECT. If this value is reset then the values *cache_hit_rate*, *cache_wait_buffer* and *nr_cache_searches* are also implicitly reset to 0.

nr_cache_searches
: Number of search operations for UTM pages in the cache taken into account to calculate the value *cache_hit_rate*.

  You can reset the value using KC_MODIFY_OBJECT. If this value is reset then the values *cache_hit_rate*, *cache_wait_buffer* and *nr_cache_rqs* are also implicitly reset to 0.

nr_res_rqs
: Number of requests for transaction resources in the last closed 100 second interval taken into account to calculate the value *wait_resources*.

nr_res_rqs_for_max
: Number of requests for transaction resources in the 100 second interval during which the maximum conflict rate *max_wait_resources* was reached.

  You can reset the value using KC_MODIFY_OBJECT. If this value is reset then the values *max_wait_resources*, *max_wait_system_resources* and *nr_sys_res_rqs_for_max* are also implicitly reset to 0.

  The values *nr_res_rqs* and *nr_res_rqs_for_max* are useful when assessing the relevance of a high lock conflict rate, in particular with regard to losses due to lock conflicts.

  Example:

  nr_res_rqs=100, *wait_resources*=5
  nr_res_rqs_for_max=10, *max_wait_resources*=50.

  I.e. the maximum lock conflict rate of 50 was reached with 10 locks being requested in 100 seconds, 5 of which led to wait times due to conflicts. In addition, the current lock conflict rate of 5 percent at 100 requested locks was also reached in100 seconds, with it again being necessary to wait for 5 locks.

nr_sys_res_rqs
: Number of requests for system resources in the last closed 100 second interval taken into account to calculate the value *wait_system_resources*

nr_sys_res_rqs_for_max
: Number of requests for system resources in the 100 second interval during which the maximum conflict rate *max_wait_system_resources* was reached.

  You can reset the value using KC_MODIFY_OBJECT. If this value is reset then the values *max_wait_resources*, *max_wait_system_resources* and *nr_res_rqs_for_max* are also implicitly reset to 0.

curr_system_tasks
: Number of UTM system processes that are currently running.

data_compression
: Specifies whether data compresion is currently enabled:

  ´Y´ Data compresion is enabled.

  ´N´ Data compresion is not enabled.

  You can modify the value with KC_MODIFY_OBJECT if data compression is permitted by means of generation (see section “kc_max_par_str - Maximum values for the application (MAX parameters)” on page 624 and openUTM manual “Generating Applications”, MAX DATA-COMPRESSION=).

  A modification applies beyond the application run; in UTM cluster applications it applies for all node applications.

avg_saved_pgs_by_compr
: Average value for the UTM pages saved per data compression. The writing of areas in which UTM performs no compression because, for example, the data length is less than one UTM page is not included in this statistics value. Two digits to the left of the decimal point and one decimal digit of the statistics value are displayed, i.e. a content of 010 corresponds to an average saving of 1.0 UTM pages.

  The value can be reset with KC_MODIFY_OBJECT.

  If no statistics values for data compression are available for the application, binary zero is output. This is possible in the following situations.

  – Data compression is disabled.
  – The value was reset with KC_MODIFY_OBJECT.
  – No data compression was performed because the application uses "small" data areas in which it does not make sense to use compression.

  If the value output for *avg_saved_pgs_by_compr* is less than 5 - which corresponds to 0.5 saved UTM pages per compression attempt -, for performance reasons data compression should be disabled for this application.

kc_diag_and_account_par_str - Diagnostic and accounting parameters

The data structure *kc_diag_and_account_par_str* is defined for the parameter type KC_DIAG_AND_ACCOUNT_PAR. *kc_diag_and_account_par_str* contains the *kc_dump_event_str* data structure, which in turn contains *kc_insert_str* .

In the case of KC_GET_OBJECT, UTM returns following information in *kc_diag_and_account_par_str*:

- which diagnostic functions are currently enabled
- if the UTM accounting is currently enabled.

You can enable and disable different diagnostic functions, the UTM event monitor KDCMON and UTM Accounting with KC_MODIFY_OBJECT and the KC_DIAG_AND_ACCOUNT_PAR parameter type.

mod[1]	Data structure kc_diag_and_account_par_str
x(GIR)	`char account;`
x(GIR)	`char calc;`
x(IR)	`char kdcmon;`
–	`char dump_msg_id[4];`
x(GIR)	`char testmode;`
x(GIR)	`char bcam_trace;`
x(GIR)	`char osi_trace;`
x(GIR)	`char osi_trace_records[5];`
x(GA)	`char sysprot_switch;`
x(GIR)	`struct kc_dump_event_str dump_event[3];`
x(IR)	`char stxit_log;`
x(IR)	`char xa_debug;`
x(IR)	`char xa_debug_out;`
–	`curr_max_btrace_lth[5];`
x(IR)	`char admi_trace;`
x(IR)	`char cpic_trace;`
x(IR)	`char tx_trace;`
x(IR)	`char xatmi_trace;`

B

mod[1]	Data structure kc_dump_event_str
x(GIR)	`char event_type[4];`
x(GIR)	`char event[4];`
x(GIR)	`struct kc_insert_str insert[3];`

mod[1]	Data structure kc_insert_str
x(GIR)	`char insert_index[2];`
x(GIR)	`union kc_value value;`
x(GIR)	`char value_type;`
x(GIR)	`char comp[2];`

[1] Field contents can be modified with KC_MODIFY_OBJECT, see page 369f.

The fields in the data structure *kc_diag_and_account_par_str* have the following meanings:

account
: Specifies if the accounting phase of the UTM accounting is enabled.

  ´Y´ The accounting phase is enabled (ON).

  ´N´ The accounting phase is disabled (OFF).

  The accounting phase can be enabled or disabled during the application run.

  For more information on UTM accounting see also the openUTM manual “Generating Applications” and the openUTM manual “Using openUTM Applications”.

calc
: Specifies if the calculation phase for the UTM accounting is enabled or disabled.

  ´Y´ The calculation phase is enabled (ON).

  ´N´ The calculation phase is disabled (OFF).

  The calculation phase can be enabled or disabled during the application run.

kdcmon
: Specifies if the UTM measurement monitor KDCMON is enabled.

  ´Y´ KDCMON is enabled (ON).

  You can evaluate the values measured by KDCMON with the UTM tool KDCEVAL. For details on operating KDCMON please refer to the relevant openUTM manual “Using openUTM Applications”.

  ´N´ KDCMON is disabled (OFF).

  KDCMON can be enabled or disabled during the application run.

dump_msg_id

This parameter is no longer supported, but is retained in the structure as a place-holder. Use the data structure *kc_dump_event_str* (see page 616).

testmode

Specifies if the test mode is enabled.
Test mode means that additional internal UTM routines are executed to conduct plausibility tests and to record internal TRACE information.

´Y´ The test mode is enabled (ON).

´N´ The test mode is disabled (OFF).

The test mode can be enabled or disabled during the application run. For performance reasons test mode should only be enabled when requested by Systems Support in order to create diagnostic documentation.

bcam_trace

Specifies if the BCAM trace is enabled. BCAM trace is the trace function which monitors all connection-specific activities within a UTM application (for example, the BCAM trace function on BS2000 systems).

´Y´ The BCAM trace function is enabled (ON).

´S´ The BCAM trace function was explicitly enabled (SELECT) for several LTERM, LPAP, MUX partners (BS2000 systems) or USERs.
Only those activities on connections to the explicitly specified LTERM, LPAP or MUX partners or user ids are logged.

´N´ The BCAM trace function is disabled (OFF).

You can enable or disable the BCAM trace function during an application run.

osi_trace

Specifies if the OSI trace function is enabled.
The OSI trace is needed for diagnosing problems with OSI TP connections of the application.

´Y´ The OSI trace function is enabled (ON). All record types are traced.

´N´ The OSI trace function is disable (OFF).

´S´ The OSI trace function is enabled for certain record types. Which record types will be traced and which will not is specified in the *osi_trace_records* field.

osi_trace is only relevant if objects for distributed processing via OSI TP have been generated in the application.
You can enable and disable the OSI trace function during the application run. For performance reasons the OSI trace should only be enabled when requested by Systems Support in order to create diagnostic documentation.

osi_trace_records

Specifies which record types will be traced in the OSI trace.
Each field element in *osi_trace_records* represents a record type:

The 1st field represents the record type "SPI",
The 2nd field represents the record type "INT"
The 3rd field represents the record type "OSS"
The 4th field represents the record type "SERV"
The 5th field represents the record type "PROT"

The entries in the individual field elements have the following meanings:

´Y´ The trace records will be recorded for the record type corresponding to the field element.

´N´ The trace records will not be recorded for the record type corresponding to the field element.

The record types have the following meanings:

SPI Events on the XAP-TP system programming interface

INT Internal program flow in an XAP-TP routine

OSS Events occurring during the processing of OSS calls (OSI session service)

SERV Internal OSS trace records of type O_TR_SERV

PROT Internal OSS trace records of type O_TR_PROT

You can enable the OSI trace during the application run for certain record types.

It is not possible to disable the trace for individual record types.

However, you can use the parameter *osi_trace='N'* disable all record types and then reactivate individual record types as required.

The contents of *osi_trace_records* is relevant if objects for distributed processing via OSI TP were generated in the application.

sysprot_switch

Specifies whether the log files in the UTM application are to be switched over.

'Y' The log files are to be switched over.

'N' The log files are not to be switched over.

B
B stxit_log

Specifies whether Stxit logging is to be enabled or disabled.

B 'Y' Stxit logging is enabled.

B 'N' Stxit logging is disabled.

B You can enable or disable Stxit logging while the application is running.

xa_debug

Specifies whether debug information for the XA connection is to be output to the database.

'Y' XA-DEBUG is enabled.
Calls of the XA interface are logged.

'A' Extended XA-DEBUG is enabled (ALL).
Specific data areas are logged in addition to the calls of the XA interface.

'N' XA-DEBUG is disabled.

You can enable or disable XA-DEBUG while the application is running.

xa_debug_out

Controls the output destinations for XA-DEBUG.

'S' Output to SYSOUT/stderr, default value.

'F' Output to a file.

If you use only the field *xa_debug* without providing a value for *xa_debug_out*, any value you specified in the start parameter .RMXA DEBUG= when starting the UTM application will be used (see openUTM manual “Using openUTM Applications”). Otherwise, the log is written to SYSOUT/stderr.

curr_max_btrace_lth

Specifies the maximum length of data that is recorded when the BCAM trace function is enabled. See also the start parameter BTRACE.

admi_trace

Specifies whether the ADMI trace function (trace function for the KDCADMI administration program interface) is enabled.
See also the start parameter ADMI-TRACE.

´Y´ The ADMI trace function is enabled.

´N´ The ADMI trace function is disabled.

You can enable or disable the ADMI trace function while the application is running.

cpic_trace

Specifies whether the CPI-C trace function (trace function for the X/Open interface CPI-C) is enabled. See also the start parameter CPIC-TRACE.

´T´ The CPI-C trace function is enabled with the level TRACE. For each function call, the content of the input and output parameters is output. Only the first 16 bytes are output from the data buffers. The return codes of the KDCS calls to which the CPI-C calls are mapped are output.

´B´ The CPI-C trace function is enabled with the level BUFFER. This trace level includes the TRACE level. However, the data buffers are logged in their full length.

´D´ The CPI-C trace function is enabled with the level DUMP. This trace level includes the TRACE level and also writes diagnostic information to the trace file.

´A´ The CPI-C trace function is enabled with the level ALL. This trace level includes the levels BUFFER, DUMP and TRACE.

´N´ The CPI-C trace function is disabled.

You can enable or disable the CPI-C trace function while the application is running.

tx_trace

Specifies whether the TX trace function (trace function for the X/Open interface TX) is enabled. See also the start parameter TX-TRACE.

´E´ The TX trace function is enabled with the level ERROR. Only errors are logged.

´I´ The TX trace function is enabled with the level INTERFACE. This trace level includes the ERROR level. TX calls are also logged.

´F´ The TX trace function is enabled with the level FULL. This trace level includes the INTERFACE level. All KDCS calls to which the TX calls are mapped are also logged.

´D´ The TX trace function is enabled with the level DEBUG. This trace level includes the FULL level and diagnostic information is also logged.

´N´ The TX trace function is disabled.

You can enable or disable the TX trace function while the application is running.

xatmi_trace

Specifies whether the XATMI trace function (trace function for the X/Open interface XATMI) is enabled. See also the start parameter XATMI-TRACE.

´E´ The XATMI trace function is enabled with the level ERROR. Only errors are logged.

´I´ The XATMI trace function is enabled with the level INTERFACE. This trace level includes the ERROR level. XATMI calls are also logged.

´F´ The XATMI trace function is enabled with the level FULL. This trace level includes the INTERFACE level. All KDCS calls to which the XATMI calls are mapped are also logged.

´D´ The XATMI trace function is enabled with the level DEBUG. This trace level includes the FULL level and diagnostic information is also logged.

´N´ The XATMI trace function is disabled.

You can enable or disable the XATMI trace function while the application is running.

dump_event

In the data structure *kc_dump_event_str*, an event is specified for which a UTM dump with an event-dependent designator is generated when the event occurs. The dump is created by the process in which the event occurred. The application is not terminated. Test mode must be enabled in order to create a UTM dump (*testmode*=´Y´). :

For detailed information, refer to section “KDCDIAG - Switch diagnostic aids on and off” on page 681.

The data structure contains a message number, a KDCS return code (KDCRCCC or KDCRCDC) or a SIGNON status code. If a message with this message number is generated or if this return code or status code is returned, a corresponding UTM dump is generated.

Description of the fields in the structure *dump_event*:

event_type

Type of event for which a UTM dump is to be generated:

'MSG' UTM message

'RCDC'
Incompatible KDCS return code

'RCCC'
Compatible KDCS return code

'SIGN'
SIGNON status code

'NONE'
Explicit deactivation of an individual event for a message dump. This allows the commands KDCDIAG DUMP-MESSAGE [1, 2 or 3] to be cancelled (see section “KDCDIAG - Switch diagnostic aids on and off” on page 681).

event Message number, KDCS return code (KDCRCCC or KDCRCDC) or SIGNON status code, depending on the *event_type*

event_type MSG
Four-digit internal message number, with leading "K" or "P", e.g. K009 or P001.

event_type RCDC
Incompatible KDCS return code: KCRCDC (4 bytes), e.g. "K301"

event_type RCCC
Three-digit compatible KDCS return code, e.g. "14Z"

event_type SIGN
SIGNON status code: KCRSIGN1 or KCRSIGN2 (3 bytes), e.g. "U01"

insert
The specifications in the data structure *kc_insert_str* only make sense for the *event_type* MSG. For detailed information, refer to section “KDCDIAG - Switch diagnostic aids on and off” on page 681.

Description of the fields in the structure *insert*:

insert_index
Number of the insert to be checked, e.g. "2" for the second insert in a message. You can specify a maximum of three inserts per message (with the structures insert[0] through insert[2]).

You can find the sequence of the inserts in a UTM message in openUTM manual ”Messages, Debugging and Diagnostics”

Possible values: 1 ... 20

To generate a message dump independently of the inserts, set all three *insert_index* values to "0".

value Value against which the insert is to be checked.

UTM represents the string in a union of the type *kc_value*.

union kc_value
`char x[64];`
`char c[32];`

For permitted values, see *value_type*.

value_type

value_type specifies how the contents of the value field are to be interpreted:

- N: numeric
- C: alphanumeric
- X: hexadecimal

comp Specifies whether the system is to test for equality or inequality. The following values are possible:

EQ Checks for equality, default.

NE Checks for inequality.

kc_dyn_par_str - Dynamic objects

The data structure *kc_dyn_par_str* is defined for the parameter type KC_DYN_PAR. In the case of KC_GET_OBJECT, UTM returns information on objects that can be created dynamically in *kc_dyn_par_str*. UTM specifies the following for the individual object types:

- The total number of objects of the object type that can be contained in the configuration.
- The number of objects of the object type that could still be added dynamically to the configuration with KC_CREATE_OBJECT.

Data structure kc_dyn_par_str

```
char lterm_total[10];
char lterm_free[10];
char pterm_total[10];
char pterm_free[10];
char program_total[10];
char program_free[10];
char tac_total[10];
char tac_free[10];
char user_total[10];
char user_free[10];
B  char card_total[10];
B  char card_free[10];
char kset_total[10];
char kset_free[10];
char ltac_total[10];
char ltac_free[10];
char queue_total[10];
char queue_free[10];
char con_total[10];
char con_free[10];
char lses_total[10];
char lses_free[10];
B  char princ_total[10];
B  char princ_free[10];
```

The fields in the data structure have the following meanings:

lterm_total
: Specifies the total number of LTERM partners that can be added to the table in the KDCFILE. *lterm_total* is also the number of table spaces generated for LTERM partners.

  The number consists of:

  – The number of statically added LTERM partners.
  – The number of dynamically added of LTERM partners (*obj_type*=KC_LTERM).
  – The number of LTERM partners of LTERM pools. The number corresponds to the sum of all NUMBER operands of TPOOL commands specified for the KDCDEF generation.
  – The number of reserved table spaces still free, i.e. in which LTERM partners can still be added.

  Deleted LTERM partners are also included in this number.

lterm_free
: Contains the number of LTERM partners that you can still add dynamically to the configuration.

pterm_total
: Specifies the total number of clients and printer that can be added to the table in the KDCFILE. *pterm_total* is also the number of table spaces generated for objects of type KC_PTERM.

  The number consists of:

  – The number of statically added clients and printers, i.e. the number of PTERM commands in the KDCDEF generation.
  – The number of dynamically added clients/printer (*obj_type*=KC_PTERM).
  – The number of connections collected in LTERM pools for clients. The number corresponds to the sum of all NUMBER operands of TPOOL commands specified for the KDCDEF generation.
  – The number of reserved table spaces still free, i.e. in which clients and printers can still be added.

  Deleted clients and printers are also contained in this number.

pterm_free
: Contains the number of clients and printers that you can still add with KC_CREATE_OBJECT.

program_total

Specifies the total number of program units that can be added to the table in the KDCFILE. *program_total* is also the number of table spaces generated for objects of type KC_PROGRAM.

The number consists of:

– The number of statically added program units and VORGANG exits, i.e. the number of PROGRAM commands in the KDCDEF generation.

– The number of dynamically added program units and VORGANG-Exits (*obj_type*=KC_PROGRAM).

– The number of reserved table spaces that are still free.

Deleted program units are also contained in this number.

program_free

Contains the number of program units and VORGANG exits that you can still add with KC_CREATE_OBJECT.

tac_total

Specifies the total number of transaction codes and TAC queues that can be added to the table in the KDCFILE. *tac_total* is also the number of table spaces generated for objects of type KC_TAC.

The number consists of:

– The number of statically added transaction codes and TAC queues, i.e. the number of TAC commands in the KDCDEF generation.

– The number of dynamically added transaction codes and TAC queues (*obj_type*=KC_TAC).

– The number of reserved table spaces that are still free.

Deleted transaction codes and TAC queues are also contained in this number.

tac_free

Contains the number of transaction codes and TAC queues that you can still add with KC_CREATE_OBJECT.

user_total

Specifies the total number of user IDs that can be added to the table in the KDCFILE. *user_total* is also the number of table spaces generated for objects of type KC_USER.

The number consists of:

– The number of statically and dynamically added user IDs (in an application generated with user IDs) or
the number of statically or dynamically added LTERM partners (in an application generated without user IDs).

– The number of clients existing with *ptype*=´APPLI´ (TS applications that are not socket applications), *ptype*=´UPIC-...´ (UPIC clients) or *ptype*=´SOCKET´ (socket applications). For these clients UTM creates internal user IDs with the name of the corresponding LTERM partner.

– The number of reserved table spaces for user IDs that are still free (in an application generated with user IDs)
or
Number of reserved table spaces for LTERM partners that are still free (in an application generated without user IDs).

Deleted user IDs are also contained in this number.

user_free

Contains the number of user IDs that you can still add dynamically with KC_CREATE_OBJECT.

B card_total

B Indicates how many user IDs with ID cards can be entered in the table in the

B KDCFILE in total. *card_total* consists of:

B – the number of statically or dynamically entered user IDs that have ID cards

B – the number of table spaces that are reserved for user IDs with ID cards and are

B still free.

B card_total

B Contains the number of user IDs with identification cards that you can still add with

B KC_CREATE_OBJECT.

kset_total

Contains the total number of key sets that can be entered in the KSET table.

kset_free

Contains the current number of key sets that you can still enter in the KSET table by means of KC_CREATE_OBJECT.

ltac_total
: Contains the total number of LTACs that can be entered in the LTAC table.

ltac_free
: Contains the current number of LTACs that you can still enter in the LTAC table by means of KC_CREATE_OBJECT.

queue_total
: Contains the total number of temporary queues that can be entered in the QUEUE table. This value was specified at generation by means of the QUEUE statement.

queue_free
: Contains the current number of temporary queues that you can still enter by means of the KDCS call QCRE.

con_total
: Contains the total number of LU6.1 transport connections that can be entered in the PTERM table.

con_free
: Contains the current number of LU6.1 transport connections that you can still enter in the PTERM table by means of KC_CREATE_OBJECT.

lses_total
: Contains the total number of LU6.1 session that can be entered in the USER table.

lses_free
: Contains the current number of LU6.1 sessions that you can still enter in the USER table by means of KC_CREATE_OBJECT.

B B princ_total
: Contains the total number of USERs created with principal.

B B princ_free
: Contains the number of USERs with principal that can still be generated.

kc_max_par_str - Maximum values for the application (MAX parameters)

The data structure *kc_max_par_str* is defined for the parameter type KC_MAX_PAR. In the case of KC_GET_OBJECT, UTM returns following information in *kc_max_par_str*:

- The basic properties of the application, for example the application name, function versions, the name of the KDCFILE.
- The maximum values for the parameters of the application, such as the size of the page pool, of the restart area and of the KDCS storage areas, the maximum number of users, the maximum number of lock codes and key codes of the application, the maximum time slice for time controlled asynchronous jobs, and the maximum number of processes that can be utilized for the application.
- The resources that will be used by the application, for example access keys for shared memory segments and semaphores.

mod[1]	Data structure kc_max_par_str	Page[2]
–	`char adf_name[16];`[3]	
–	`char applimode;`	627
–	`char appliname[8];`	627
–	`char asyntasks[3];`	627
–	`char blksize[2];`	627
x(GIR)	`char bretrynr[5];`	628
–	`char cacheshmkey[10];`	628
–	`char cachesize_pages[10];`	628
x(GIR)	`char cachesize_paging[3];`	628
–	`char cachesize_res;`	628
–	`char cardlth[3];`	629
–	`char catid_a[4];`	629
–	`char catid_b[4];`	629
–	`union kc_clear_char clrch;`	629
–	`char clrch_type;`	629
x(IR)	`char conn_users[10];`	630
x(GPD)	`char destadm[8];`	630
–	`char dputlimit1_day[3];`	630
–	`char dputlimit1_hour[2];`	630
–	`char dputlimit1_min[2];`	630
–	`char dputlimit1_sec[2];`	630

	mod[1]	Data structure kc_max_par_str	Page[2]
	–	char dputlimit2_day[3];	631
	–	char dputlimit2_hour[2];	631
	–	char dputlimit2_min[2];	631
	–	char dputlimit2_sec[2];	631
	–	char gssbs[10];	631
	–	char hostname[8];	631
X/W	–	char ipcshmkey[10];	631
X/W	–	char ipctrace[10];	631
X/W	–	char kaashmkey[10];	631
	–	char kb[10];	631
	–	char kdcfile_name[42];	632
	–	char kdcfile_operation;	632
	–	char keyvalue[4];	632
B	–	char locale_lang_id[2];	632
B	–	char locale_terr_id[2];	632
B	–	char locale_ccsname[8];	632
	–	char lputbuf[4];	633
	–	char lputlth[10];	633
	–	char lssbs[4];	633
	–	char mp_wait_sec[5]	633
	–	char nb[10];	633
X/W	–	char net_access;	633
	–	char nrconv[2];	633
	–	char osi_scratch_area[5];	633
X/W	–	char osishmkey[10];	634
	–	char pgpool_pages[10];	634
	–	char pgpool_warnlevel1[2];	634
	–	char pgpool_warnlevel2[3];	634
	–	char pgpoolfs[5];	634
X	–	char pisizelth[5];	634
	–	char recbuf_pages[10];	634
	–	char recbuf_lth[10];	634
	–	char recbuffs[3];	635

	mod[1]	Data structure kc_max_par_str	Page[2]
B	–	`char reqnr[3];`	635
	–	`char seclev`[4]`;`	
B	–	`char sat;`	635
X/W	–	`char semarray_startkey[10];`	635
X/W	–	`char semarray_number[4];`	635
X/W	–	`char semkey[10][10];`	636
	–	`char signon_value[3];`	636
	–	`char signon_restr;`	636
	x (GIR)	`char signon_fail[3];`	636
	x (GIR)	`char sm2;`	636
	–	`char spab[10];`	637
	–	`char syslog_size[10];`	637
	–	`char tasks[3];`	637
	–	`char tasks_in_pgwt[3];`	637
	–	`char tracerec[5];`	638
	–	`char trmsglth[10];`	638
	–	`char uslog;`	638
B	–	`char vgmsize[3];`	638
X/W	–	`char xaptpshmkey[10];`	638
	–	`char mpgpool_pages[10];` [4]	
	–	`char mpgpool_res;` [4]	
	–	`char rtimer;` [4]	
	–	`char spin_lock_asyn[10];` [4]	
	–	`char spin_lock_cache[10];` [4]	
	–	`char spin_lock_kaa[10];` [4]	
	–	`char spin_lock_ipc[10];` [4]	
	–	`char spin_lock_pcmm[10];` [4]	
	–	`char xopen_cpic_dspl[5];` [4]	
	–	`char xopen_cpic_lth[5];` [4]	
	–	`char xopen_xatmi_dspl[5];` [4]	
	–	`char xopen_xatmi_lth[5];` [4]	
	–	`char xopen_tx_dspl[5];` [4]	
	–	`char xopen_tx_lth[5];` [4]	

mod[1]	Data structure kc_max_par_str	Page[2]
–	char max_statistics_msg;	638
–	char max_open_asyn_conv[10];	638
–	char dead_letter_q_alarm[10];	639
–	char max_suspended_ta[3]; [4]	
–	char atac_redelivery[3];	639
–	char dget_redelivery[3];	639
–	char principal_lth[3];	638
–	char privileged_lterm[8];	639
	char cache_location;	639
	char data_compression;	639

B

[1] Field contents of the field can be modified with KC_MODIFY_OBJECT; see page 377

[2] The meaning of the field is described on the page specified in this column.

[3] Filled with blanks by default.

[4] Internal UTM field; the contents of this field are irrelevant and will not be described in the following.

The fields in the data structure have the following meanings:

applimode
: Specifies if the UTM application is a UTM-S or UTM-F application.

  ´S´ The application was generated as a UTM-S application (Secure).

  ´F´ The application was generated as a UTM-F application (Fast).

appliname
: The name of the UTM application. This name is defined in MAX APPLINAME during the static generation with the KDCDEF generation tool.

  appliname is the name of the application that must be specified by terminals when establishing a connection.

asyntasks
: Contains the maximum number of processes of the application that may process jobs to asynchronous transaction codes. *asyntasks* is the upper limit for the current number of processes used for processing asynchronous jobs. The value can be set at the start of the application or it can be set dynamically by the administration.

blksize
: Specifies the size of a UTM page. The size is set during the KDCDEF generation to either 2K, 4K or 8K. Possible values:

  ´2´ The size of a UTM page is 2K.

´4´ The size of a UTM page is 4K.
´8´ The size of a UTM page is 8K.

B bretrynr

B Contains the number of times UTM will attempt to pass a message to the transport
B system (BCAM) when BCAM cannot immediately accept the message at the
B present time. If the value of *bretrynr* is exceeded the connection to the dialog partner
B is closed down. The value of *bretrynr* influences the performance of the application.

B For asynchronous messages sent to a dialog partner with *ptype*= ´APPLI´ (TS appli-
B cations that are not socket applications), *bretrynr* ≥ 3 means that UTM will try to pass
B the message on to BCAM up to three times. If BCAM does not accept the message
B on the third try, then UTM will release the process for now, but will not close the
B connection. After a 3 second wait UTM will try again up to three times to pass the
B message to BCAM. If the attempts fail again, then UTM waits another 3 seconds
B before trying another three times, etc.

B Minimum value: ´1´
B Maximum value: ´32767´ (theoretical value)

X/W cacheshmkey

X/W Contains the access key for the shared memory segment that contains the global
X/W application buffer for file accesses. *cacheshmkey* is a global parameter for Unix, Linux
X/W and Windows systems. *cacheshmkey* is a decimal number.

cachesize_pages

Specifies the size of the cache in UTM pages. The size of a UTM page is returned in *blksize*. All access to the page pool is carried out via the cache, i.e. all input and output to local secondary storage areas, global secondary storage areas, terminal-specific long-term storage area, LPUT and FPUT messages, MPUT messages, as well as some UTM administration data. A write to a KDCFILE is only executed if there is no more space in the cache or if the transaction is terminated.

cachesize_paging

Specifies the percentage of the cache that will be written at one time to the KDCFILE when a bottleneck occurs so that the storage space in the cache can be used for other data. The value of *cachesize_paging* influences the performance of your UTM application.

UTM removes at least 8 UTM pages from the cache when paging even if the value of *cachesize_paging* is less than this number of UTM pages.

Minimum value: ´0´, i.e. 8 UTM pages will be removed for storage elsewhere
Maximum value: ´100´ (%)

B cachesize_res

B Specifies whether or not the cache is resident. The contents of the field are to be
B interpreted as follows:

B ´R´ The cache is resident.

B ´N´ The cache is pageable, i.e. not resident.

cardlth

The length in bytes of the identification information that UTM stores when an ID reader is used in addition to the access privilege check done when signing on (KDCSIGN). The identification information can be read in a program unit using the KDCS call INFO.

B catid_a

B Contains the catalog ID (CAT-ID) assigned to your KDCFILE with the suffix A for the

B B2000 system.

B catid_b

B The *catid_b* is only relevant if you maintain a redundant copy of the KDCFILE.

B *catid_b* then contains the catalog ID (CAT-ID) assigned to your KDCFILE with the

B suffix B. If only one KDCFILE is used, then *catid_b* = *catid_a*.

clrch (**cl**ea**r ch**aracter)

Contains the character with which the communication area (KB) and the standard primary working area (SPAB) of the program units are overwritten at the end of a dialog step.

If no character was defined during generation, then *clrch* contains blanks and *clrch_type*=´N´. The storage areas are not overwritten then at the end of a dialog step.

If a character was defined in the KDCDEF generation, then *clrch* contains one character. If the character is hexadecimal, then each half byte is represented as one character.

clrch is returned in the form of the following union:

union kc_clear_char
`char x[2];`
`char c;`

The *x* field contains data if *clrch* is returned as a hexadecimal character.
The *c* field contains data if *clrch* is returned as an alphanumeric character.

You can determine how to interpret the data contained in *clrch* using the *clrch_type* field.

clrch_type

Specifies how the contents of the *clrch* field are to be interpreted. The contents mean:

´X´ *clrch* contains a hexadecimal character.

´C´ *clrch* contains a printable, alphanumeric character.

´N´ No *clrch* character is defined.

conn_users

The maximum number of users that may be signed on to the UTM application at the same time. Users are understood as being the number of user IDs that may be signed on at the same time. If the application is generated without user IDs, then the number of clients that can connect to the application via LTERM partners is limited by *conn_users*.

User IDs generated with administration privileges can still sign on to the UTM application if the maximum number of simultaneously active user IDs has already been reached.

conn_users=´0´ means that the number of simultaneously active users is unlimited.

Minimum value: ´0´
Maximum value: ´500000´

X/W X/W X/W When performing modifications on Unix, Linux and Windows systems, no value greater than the value defined during the generation may be specified (MAX CONN-USERS) .

destadm

Contains the receiver to which UTM sends the results of KDCADM administration calls that were asynchronously processed (asynchronous transaction codes from KDCADM). The receiver can be an LTERM partner or an asynchronous TAC or a TAC queue.

If *destadm* contains blanks, then no receiver is defined. The results of the asynchronous transaction codes from KDCADM are lost. In this case you are to define a receiver using, for example, KC_MODIFY_OBJECT.

dputlimit1_day, dputlimit1_hour, dputlimit1_min, dputlimit1_sec

These parameters determine the upper limit of the time interval in which a time controlled job must be executed. Time controlled jobs are created with the KDCS call DPUT. A program unit call and hence also a DPUT call with an absolute time specification can be delayed to such an extent that the required execution time of the DPUT has already elapsed. This time at which a time controlled job is to be executed (specified in the DPUT call) must occur within the time span specified in *dputlimit1* **after** the time of the DPUT call. *dputlimit1* is specified as follows:

The number of days (*dputlimit1_day*) + the number of hours (*dputlimit1_hour*) + the number of minutes (*dputlimit1_min*) + the number of seconds (*dputlimit1_sec*). Therefore, the following is true:

Execution time < time of the DPUT call + *dputlimit1*

dputlimit2_day, dputlimit2_hour, dputlimit2_min, dputlimit2_sec

These parameters determine the lower limit of the time interval in which a time controlled job (DPUT call) must be executed. The time controlled job is to be executed (specified in the DPUT call) no earlier than within the time span specified in *dputlimit1* **before** the time of the DPUT call. *dputlimit2* is specified as follows:

The number of days (*dputlimit2_day*) + the number of hours (*dputlimit2_hour*) + the number of minutes (*dputlimit2_min*) + the number of seconds (*dputlimit2_sec*).

Therefore, the following is true:
Execution time > time of the DPUT call - *dputlimit2*

If the execution time specified lies between the limit specified in *dputlimit2* and the time of the call, then the DPUT is immediately converted to an FPUT.

gssbs The maximum number of global secondary storage areas that may exist in the application at one time.

hostname

Contains the name of the virtual host on which (from BCAMs point of view) the application is running.

X/W ipcshmkey

X/W Contains the access key for the shared memory segment used for interprocess communication between the work processes on the one hand, and the external processes of the application on the other hand. On Unix, Linux and Windows systems *ipcshmkey* is a global parameter. *ipcshmkey* is a decimal number.

X/W ipctrace

X/W Contains the number of entries in the trace area of the IPC.
X/W UTM writes the entries into the trace area of the IPC (shared memory segment for the interprocess communication) if the UTM application is running in test mode (TESTMODE=ON). These entries contain internal information for diagnostic purposes. One entry takes up 32 bytes. If the number of entries contained in *ipctrace* is exceeded, then UTM overwrites already existing entries, starting with the oldest entry.

X/W kaashmkey

X/W Contains the access key for the shared memory segment in which the global application data is stored. *kaashmkey* is a global parameter on Unix, Linux and Windows systems. *kaashmkey* contains a decimal number.

kb Contains the length of the communication area in bytes. The communication area header and the communication area return area are not taken into consideration when determining this length.

kdcfile_name

Base name of the KDCFILE, USLOG user log file and the SYSLOG system log file (see also the openUTM manual “Generating Applications”). *kdcfile_name* must also be specified for the start of the application in the FILEBASE start parameter.

kdcfile_operation

Specifies if a redundant copy of the KDCFILE is maintained or not. The contents of *kdcfile_operation* are interpreted as follows:

´D´ A redundant copy of the KDCFILE is maintained. If the KDCFILE is split (see also the openUTM manual “Generating Applications”, KDCFILE), then all KDCFILE files will be maintained together with a redundant copy.

´S´ Only one copy of the KDCFILE is maintained. If the KDCFILE was split, then only one copy of each KDCFILE file is maintained.

keyvalue

Contains the number of the highest key code in the application and therefore the number of the highest lock code that may be used for access protection for a transaction code or an LTERM partner.

keyvalue also specifies the maximum number of key codes per key set.

B locale_lang_id, locale_terr_id, locale_ccsname

B These contain the three components of the locale assigned to the UTM application. The locale defines the standard language environment of the application. The standard language environment is assigned to every user ID (KC_USER), every LTERM partner and every LTERM pool of the application as the standard setting for the language environment. The standard setting is in effect as long as a locale is not defined for these objects (see also the openUTM manual “Generating Applications”).

B *locale_lang_id*

B Contains the up to two characters long language code.

B *locale_terr_id*

B Contains an up to two characters long territorial code.

B *locale_ccsname*

B (**c**oded **c**haracter **s**et **name**)

B Contains the up to 8 characters long name of an expanded character set (CCS name; see also the XHCS User Guide).

lputbuf Contains the size of the buffer in which UTM temporarily stores the records created with the KDCS call LPUT before it writes them to the user log file (USLOG). The buffer is stored in the page pool.

The LPUT statements created in the program units are temporarily stored in this buffer until it is full. Only then will UTM copy the statements into the user log file. The user log file (USLOG) is only open during this copy procedure.

lputlth Contains the maximum length of the user data in an LPUT record.
The length of an LPUT record consists of:
lputlth + 84 bytes for the communication area header + 12 bytes for the length fields.

lssbs Contains the maximum number of LSSBs (local secondary storage areas) that can be created within a service.

mp_wait_sec (**m**emory **p**ool **wait**)
Specifies the maximum number of seconds a UTM application program will wait to connect a process to a common memory pool.

nb (KDCS message area)
Contains the maximum length of the message area for KDCS program units.

X/W net_access
X/W Specifies if the connection of the application to the network.

X/W ´M´ This means that several network connections are administered in one net
X/W process. The assignment of a network connection to a net process is done
X/W via listener IDs. All connections with the same listener ID are administered
X/W by the same net process. The listener IDs are assigned the BCAMAPPL
X/W names and access points of the local application during the KDCDEF
X/W generation.

nrconv (**n**umbe**r** of **conv**ersations)
The maximum number of services that a user may have on the stack at the same time. The value ´0´ means that no services may be placed on the stack.

osi_scratch_area
The size of an internal UTM working area that UTM needs for dynamically storing data when the OSI TP protocol is used. The number is specified in kilobytes.

B In UTM applications on BS2000 systems this working area is automatically
B increased in size, if necessary, during the application run.

X/W In UTM applications on Unix, Linux or Windows systems the size of the internal
X/W working area is constant during the entire application run. If the size of the internal
X/W working area is determined to be insufficient during operations, then the KDCDEF
X/W generation must be repeated using a higher value.

X/W
X/W
X/W
X/W
osishmkey
: Contains the access key for the shared memory segment used by OSS for the communication via OSI TP. *osishmkey* is a global parameter on Unix, Linux and Windows systems. *osishmkey* is a decimal number.

pgpool_pages
: Specifies the size of the page pool as a number of UTM pages. The size of a UTM page is output in the *blksize* field.

pgpool_warnlevel1, pgpool_warnlevel2
: Contains the number of warning levels used by UTM to warn of an impending overrun of the page pool.

  pgpool_warnlevel1
  Specifies how full the page pool must be before UTM outputs the first warning (UTM message K041). *pgpool_warnlevel1* is a decimal number in percent.

  pgpool_warnlevel2
  Specifies how full the page pool must be before UTM outputs the second warning. Asynchronous jobs are rejected after the value for warning level 2 is exceeded. In this case, the user receives the UTM message K041, and a program unit receives the corresponding return code. *pgpool_warnlevel2* is a decimal number in percent.

pgpoolfs
: Contains the number of files over which the page pool is divided. If *pgpoolfs* = ´0´, then the page pool is stored in the main file of the KDCFILE, i.e. the page pool was not swapped out.

  In the case of dual opeation of the KDCFILE the page pool in the second KDCFILE also consists of *pgpoolfs* files.

X
X
pisizelth
: Length of the restart area of the formatting system in bytes.

recbuf_pages
: Contains the size of the restart area per process. The size is specified as a number of UTM pages. The size of a UTM page is output in the *blksize* field.

  The data needed for the restart after a system error is written to the restart area. *recbuf_pages* influences the performance of the application: if this area is large, then the load placed on the running application is lower; a restart after a system error takes longer, however. If the area is small, then the load placed on the running application is higher, but a restart is faster.

recbuf_lth
: Contains the size of the buffer in bytes available per process of the application for temporarily storing restart data. The data is needed to execute a restart after a transaction or system error.

recbuffs

Contains the number of files over which the restart area is divided.

If *recbuffs* = ´0´, then the restart area is stored in the main file of the KDCFILE, i.e. the restart area was not swapped out.

In the case of dual opeation of the KDCFILE the restart area in the second KDCFILE also consists of *recbuffs* files.

B reqnr Contains the maximum number of PAM read/write jobs that may be accepted at one time in a UTM process for a file. *reqnr* contains the value set in the KDCDEF generation as long as this value is smaller than the value of *cachesize_pages*. If the value generated is larger, then the value of *cachesize_pages* is output for *reqnr*.

B sat (**s**ecurity **a**udit **t**rail)
B Specifies if SAT logging is enabled for the application.

B The SAT logging can be enabled and disabled using the KDCMSAT transaction code (see the openUTM manual “Using openUTM Applications on BS2000 Systems”, UTM-SAT administration).

B ´Y´ The SAT logging is enabled (ON).

B ´N´ The SAT logging is disabled (OFF).
B The only events logged are KDCMSAT transaction code accesses (except for KDCMSAT HELP). All other events are not logged.

X/W semarray_startkey, semarray_number
X/W Specifies the area for keys for the global application semaphores. Semaphores are used for process synchronization. The keys are global parameters on Unix, Linux and Windows systems.

X/W *semarray_startkey*
X/W Contains the number of the first semaphore key.

X/W *semarray_number*
X/W Contains the number of keys currently being used by the application.

X/W UTM uses a key by adding 1 to the *semarray_startkey* key each time, starting with
X/W the first key number.

X/W If there was no key area defined in the KDCDEF generation, then UTM returns the
X/W value ´0´ in *semarray_startkey* and *semarray_number*. In this case, UTM returns the
X/W semaphore key in the *semkey* field.

X/W semkey (**sem**aphore **key**)
X/W If UTM returns values in the *semarray_startkey* and *semarray_number* fields that are
X/W not equal to ´0´, then *semkey* contains ´0´. If the *semarray_startkey* and
X/W *semarray_number* fields contain ´0´, then *semkey* contains the key of the application
X/W for all semaphores that are global to the application (process synchronization). The
X/W keys are global parameters on Unix, Linux and Windows systems. The keys are
X/W specified as decimal numbers. A maximum of 10 keys are returned. If less than 10
X/W keys were generated, then the rest of the field contains ´0´.

signon_value
Specifies the percentage of user IDs that may have a sign on service active at one time. UTM attempts to obtain the necessary resources according to this number (see the section “kc_signon_str - Properties of the sign-on process” on page 644).

signon_restr
Specifies if restrictions were generated for the sign-on procedure (see also the section “kc_signon_str - Properties of the sign-on process” on page 644):

´R´ Database calls and access to the global UTM storage area are not permitted during the first part of the sign-on procedure (RESTRICTED).

´N´ Database calls and access to global UTM storage are permitted during the first part of the sign-on procedure.

signon_fail
Specifies the number of unsuccessful sign-on attempts repeated by a terminal user without interruption after which UTM should trigger a "silent alarm". In the case of a silent alarm, UTM generates the message K094, writes this to SYSLOG and possibly also outputs it at other message destinations configured for this message. See also the section “kc_signon_str - Properties of the sign-on process” on page 644.

Minimum value: ´1´
Maximum value: ´100´

sm2 Specifies if the UTM application sends performance data to openSM2 for monitoring.

´0´ Performance monitoring using openSM2 is generally not permitted for the UTM application. This means that the UTM application may not send any data to openSM2. The sending of data to openSM2 cannot be enabled by the administration, either.

´N´ The UTM application may send data to openSM2. The sending of data to openSM2 is currently disabled, however. It can be enabled by the administration.

´Y´ The UTM application may send data to openSM2. The sending of data to openSM2 is enabled. It can be disabled by the administration.

spab Contains the maximum length of the standard primary working area (SPAB).

syslog_size
Contains the generated control value setting used by UTM for the automatic monitoring of the size of the SYSLOG file. The automatic monitoring of the size of the SYSLOG is only possible if the SYSLOG was created as a file generation group (FGG) or a file generation directory (see the openUTM manual “Using openUTM Applications”). UTM switches to the next file generation of the SYSLOG FGG when the size of the file generation currently being written to reaches the *syslog_size* control value.

syslog_size = ´0´ means that UTM does not monitor the size of the SYSLOG file. UTM writes all UTM messages with a SYSLOG message line into this file generation.

syslog_size = ´0´ is always output when a redundant copy of the SYSLOG file is not maintained.

You can switch the logging to another file generation, change the control value or enable/disable the monitoring of the size (see KC_SYSLOG page 422 or KDCSLOG page 779)

tasks The maximum number of processes that may be used for the application at one time. *tasks* contains the maximum value set using KDCDEF (in MAX TASKS).

The number of processes that may process jobs of the application is reset at every start of the application and can be adjusted according to the current demands during the application run (see KDCAPPL page 666 and KC_MODIFY_OBJECT page 380). However, neither the number of processes specified at the start nor the number set by the administration may exceed the value returned in *tasks*.

tasks_in_pgwt
Specifies the maximum number of processes that may simultaneously process jobs with blocking calls such as the KDCS call PGWT (Program Wait).

The current setting for the number of processes is returned in *kc_tasks_par_str* when an information query with the KC_TASKS_PAR parameter type is sent.

The current number of processes is set at the start of the application and can be altered by the administration when bottlenecks arise (see KDCAPPL page 666 and KC_MODIFY_OBJECT page 380). Neither the number of processes specified at the start nor the number set by the administration may exceed the value returned here.

If *tasks_in_pgwt*=´0´, no blocking calls are allowed.

tracerec (**trace rec**ords)
: Contains the maximum number of entries in the TRACE area. UTM writes diagnostic information to this area if TESTMODE=ON has been set.
Each entry is 64 bytes long on 32-bit platforms and 128 bytes on 64-bit pattforms.

trmsglth (**tr**ansfer **m**e**s**sa**g**e **l**eng**th**)
: Contains the maximum length of the physical messages exchanged between clients, partner applications or printers and the UTM application. Control characters, position data, etc., is included in this length specification. A 64-byte internal UTM header is also contained in this number. The number is specified in bytes.

uslog Specifies if a redundant copy of the user log file is maintained for data security reasons.

´S´ (SINGLE)
Only one copy of the user log file is maintained.

´D´ (DOUBLE)
A redundant copy of the user log file is maintained as well.

For more information on the user log file consult the openUTM manual “Using openUTM Applications”.

B B B B vgmsize
: Contains the size of the buffer used for storing transaction and procedure information of an SQL database system. This will also limit the size of a user’s portion of the page pool. *vgmsize* is specified in KB.

X/W X/W X/W xaptpshmkey
: Contains the access key for the shared memory segment used by XAPTP for the communication via OSI TP.

X/W X/W
: *xaptpshmkey* is a global parameter on Unix, Linux and Windows systems.
xaptpshmkey is a decimal number.

max_statistics_msg
: Indicates whether or not the application generates statistics message K081 every hour (see the openUTM manual ”Messages, Debugging and Diagnostics” for K081, and the openUTM manual “Generating Applications” for MAX STATISTICS-MSG).

´Y´ Statistics message K081 is generated every hour and written into the SYSLOG.
When the message is issued, various application-specific statistics values are reset to zero.

´N´ Statistics message K081 is not generated.
The application-specific statistics values can be reset with the administration functions, if necessary (see KC_MODIFY_OBJECT, KC_CURR_PAR on page 365).

max_open_asyn_conv
: Contains the maximum number of asynchronous processes that can be active simultaneously.

dead_letter_q_alarm
: Controls monitoring of the number of messages in the dead letter queue. Message K134 is output each time the threshold is reached.
Monitoring is disabled if a threshold value of 0 is specified.

atac_redelivery
: Contains the maximum number of repeated deliveries of a message to an asynchronous service when the service is terminated abnormally.

dget_redelivery
: Contains the maximum number of repeated deliveries of a message to a service-controlled queue when rolling back the transaction.

B B B principal_lth
: Contains the maximum length of a Kerberos principal in bytes (see openUTM manual “Generating Applications”, MAX PRINCIPAL-LTH=).

privileged_lterm
: Contains the name of the privileged LTERM (see openUTM manual “Generating Applications”, MAX PRIVILEGED-LTERM=).

cache_location
: Returns the storage location of the UTM cache.

´P´ The UTM cache is created in the program space.

X/W X/W For Unix, Linux, and Windows systems. the value ´P´ is always returned here.

B B ´D´ The UTM cache is created in one or more data spaces (see openUTM manual “Generating Applications”, MAX CACHESIZE=).

data_compression
: Specifies whether data compresion is permitted via generation:

´Y´ Data compresion is permitted.

´N´ Data compresion is not permitted

See openUTM manual “Generating Applications”, KDCDEF statement DATA-COMPRESSION=.

kc_msg_dest_par_str - Properties of the user-specific message destinations

The *kc_msg_dest_all_par_str* data structure is defined for the KC_MSG_DEST_PAR object type. This data structure contains the four structures *user_dest_1*, *user_dest_2*, *user_dest_3* and *user_dest_4* in which, in the case of KC_GET_OBJECT, UTM provides the information on the four user-specific message destinations.

If a message destination is not generated, blanks are returned.

Data structure kc_msg_dest_all_par_str

```
struct kc_msg_dest_par_str  user_dest_1;
struct kc_msg_dest_par_str  user_dest_2;
struct kc_msg_dest_par_str  user_dest_3;
struct kc_msg_dest_par_str  user_dest_4;
```

where

Data structure kc_msg_dest_par_str

```
char md_name[8];
char md_type;
char md_format;
```

The fields of the data structure have the following meanings:

md_name
: Contains the name of the user-specific message destination.

md_type
: Specifies the type of the message destination in *name*. Possible values are:

  ´L´ for an LTERM partner

  ´T´ for a TAC or a TAC queue

  ´U´ for a user ID or a USER queue

md_format
: Indicates the format in which messages are passed to the message destination. Possible values are:

  ´F´ (FILE)
  The format corresponds to the data structures for the MSGTAC program (see the section “Control using the MSGTAC program” on page 148).

  ´P´ (PRINT)
  The format corresponds to the output format of the UTM tool KDCPSYSL (see the openUTM manual “Using openUTM Applications”).

kc_pagepool_str - Current utilization of the page pool

The data structure *kc_pagepool_str* is defined for the parameter type KC_PAGEPOOL. In the case of KC_GET_OBJECT, UTM returns information on the current utilization of the page pool in *kc_pagepool_str*.

Data structure kc_pagepool_str

```
char total_pages[10];
char free_pages[10];
char gssb_pages[10];
char lssb_pages[10];
char tls_pages[10];
char uls_pages[10];
char dial_conv_pages[10];
char tacclass_pages[10];
char fpmm_pages[10];
char fput_pages[10];
char msgtac_pages[10];
char lput_pages[10];
char phys_msg_pages[10];
char reset_msg_pages[10];
char log_rec_pages[10];
char other_pages[10];
```

The fields in the data structure have the following meanings:

total_pages
: Total number of pages in the page pool.

free_pages
: Number of free pages.

gssb_pages
: Number of pages which are utilized for GSSBs.

lssb_pages
: Number of pages which are utilized for LSSBs.

tls_pages
: Number of pages which are utilized for TLS areas.

uls_pages
: Number of pages which are utilized for ULS areas.

dial_conv_pages
: Number of pages which are utilized for service contexts by users.

tacclass_pages
: Number of pages which are utilized for dialog input messages, and which are temporarily stored in TAC Class Queues.

fpmm_pages
: Number of pages which are required for managing asynchronous messages.

fput_pages
: Number of pages which are utilized for asynchronous messages.

msgtac_pages
: Number of pages which are utilized for MSGTAC messages.

lput_pages
: Number of pages which are utilized for temporarily stored LPUT records.

phys_msg_pages
: Number of pages which are utilized for output messages and which need to be temporarily stored because they can only be transferred to the transport system in sections owing to their length.

reset_msg_pages
: Number of pages which are utilized for reset messages.

log_rec_pages
: Number of pages which are utilized for OSI TP log records.

other_pages
: Number of other utilized pages.

In the case of UTM cluster applications, GSSB and ULS areas are stored in the global page pool of the UTM cluster application. As KC_PAGEPOOL only displays the utilization of the local page pool, the values for *gssb_pages* and *uls_pages* are always zero in UTM cluster applications.

kc_queue_par_str - Properties of queue objects

The *kc_queue_par_str* data structure is defined for the KC_QUEUE_PAR parameter type. In the case of KC_GET_OBJECT, UTM returns general information on temporary queues in *kc_queue_par_str*.

Data structure kc_queue_par_str
`char qp_number[10];`
`char qlev[5];`
`char qmode;`

The fields of the data structure have the following meanings:

qp_number
: Generated maximum number of queue objects that can exist at any one time during an application run

qlev
: Default value when a temporary queue is created:
The maximum number of messages that can be in a temporary queue at any one time

qmode
: Default value when a temporary queue is created:
Response of UTM when the maximum permitted number of messages in the queue is exceeded. Possible values are:

  ´S´ (STD)
  : UTM rejects any further messages for this queue.

  ´W´ (WRAPAROUND)
  : UTM accepts further messages. When a new message is entered, the oldest message in the queue is deleted.

kc_signon_str - Properties of the sign-on process

The data structure *kc_signon_str* is defined for the object type KC_SIGNON. In the case of KC_GET_OBJECT, UTM returns the values of the parameters through which the communication partner is signed on to the application in *kc_signon_str*.

Data structure kc_signon_str

```
char concurrent_terminal_signon[3];
char grace;
char pw_history[2];
char restricted;
char silent_alarm[3];
char upic;
char multi_signon;
char omit_upic_signoff;
```

The fields of the data structure have the following meanings:

concurrent_terminal_signon
: Only relevant if a sign-on process is generated in your application.

  concurrent_terminal_signon specifies in percent for how many of the generated users the sign-on process which has been started for a sign-on via a terminal or a TS application (APPLI or SOCKET) can be active at one time.

  UTM tries to make available the required resources according to this value.

grace (Grace-Sign-On)
: Specifies whether a user may still change the password when first signing on after the password has expired (see *kc_user_str.protect_pw_time*).

  ´N´ The user cannot change the password after it has expired. Only the administrator can do this.

  ´Y´ The user can still change the password after it has expired.
  The modification must be made within the sing-on before the user is entirely signed on.

  If a sign-on service is activated, the password can be changed there using the KDCS call SIGN CP, regardless of the client type. A sign-on service is always activated when a user signs on via a connection for whose transport access point a sign-on service has been generated.

The table below shows how the individual client types behave when a password has expired and how this behavior depends on whether a sign-on service is activated.

Client type	Behavior if the password has expired[1)]
UPIC	Regardless of whether a sign-on service is activated, the password can be changed using the *Set_Conversation_Security_New_Password* function.
BS2000 terminal	If the password is blanked out, openUTM prompts the user to change the password, regardless of whether a sign-on service is activated.
	If the password is not blanked out, openUTM prompts the user to change the password only if no sign-on service is activated.
Terminal on Unix, Linux and Windows systems	openUTM prompts the user to change the password, regardless of whether a sign-on service is activated.
TS application	The user can no longer change the password without activation of a sign-on service.

1) The password can always be changed via the administration interface (e.g. KC_MODIFY_OBJECT, *obj_type*=KC_USER). By default, passwords with limited periods of validity are immediately set to "expired" when changes are made via the administration interface. If you want to prevent this, then you must explicitly request this in the administration interface.

pw_history
: Specifies for how many password changes per user UTM records a password history. *pw_history* contains the number of passwords of each user ID which UTM records.

  If a user changes the password and if a limited validity period is generated for the password in the USER statement, the new password must differ from the current password and from the last *n* passwords set for that user ID. *n* is the number in *pw_history*.

  pw_history=0 means that UTM does not keep a password history.

  The password history is only relevant when a password is changed by the user; the administrator can change the password irrespective of the passwords contained in the history.

restricted
: Specifies whether database calls and accessing global UTM Sorage areas are not allowed in the first part of the sign-on.

´Y´ Database calls and accessing global UTM storage areas are not allowed in the first part of the sign-on.

´N´ Database calls and accessing global UTM storage areas are allowed in the first part of the sign-on.

silent_alarm

Specifies after how many unsuccessful attempts of a terminal user to sign on UTM issues a silent alarm. Silent alarm means that UTM issues message K094.
This value can be modified in the *signon_fail* field in the data structure *kc_max_par_str*, see page 636.

upic Only relevant if an sign-on process was generated in your application.
upic specifies whether the sign-on process is activated when an UPIC client wishes to start a conversation.

´Y´ If a sign-on process is generated for the transport system access point by means of which the UPIC client has set up the connection, this is started before every conversation initiated by the UPIC client.

´N´ No sign-on process is started for UPIC clients.

multi_signon

Specifies whether several users can be signed on with the application under the same user ID at the same time.

´Y´ The following cases must be distinguished:

– The user ID is generated with RESTART=NO:

  Several users can be signed on with the application under the same user ID at the same time. However, only one of the users may be signed on at the terminal.

– The user ID is generated with RESTART=YES:

  Several job-receiving services can only be active under the same user ID at the same time if the job-receiving services are started via OSI TP connection and the “commit” function is selected.

´N´ No more than one user can be signed on with each user ID in the application, i.e. no more than one dialog service may be active per user ID and, if a user is signed on with the application, then no job-receiving service can be started for this specific user ID.

multi_signon has no effect on issuing asynchronous serviced through an OSI TP connection.

omit_upic_signoff

Specifies whether or not the user ID under which a UPIC client program has signed on continues to be signed on after a UPIC conversation has finished.

´Y´ The user ID continues to be signed on after the end of a UPIC conversation. This user is only signed off again
- if the connection is cleared or
- if a UPIC client with another user ID wants to sign on via this connection.

If the UPIC client does not send another user ID then the original user ID continues to be signed on, i.e. no sign-on service is started before the start of the new UPIC conversation.

In the case of applications without a user ID, a sign-on service may, if necessary, be started once after the establishment of the connection and before the start of the first UPIC conversation.

Default in UTM cluster applications.

´N The user ID with which a UPIC client has signed on is signed off after the end of each UPIC conversation.

Default in standalone UTM applications.

kc_system_par_str - System parameters

The data structure *kc_system_par_str* is defined for the parameter type KC_SYSTEM_PAR. In the case of KC_GET_OBJECT, UTM returns following information in *kc_system_par_str*:

- The basic settings of the application, for example if the application is generated for server-server communication.
- The UTM version together with its update information.
- The application name and functionality.
- The operating system and the name, platform and operating mode of the computer on which the application runs.

Data structure kc_system_par_str

```
char appliname[8];
char utm_version[8];
char applimode;
char system_type;
char hostname[8];
char destadm[8];
char tacclasses;
char pgwt;
B  char kdcload;
char load_module_gen;
char prog_change_running;
char inverse_kdcdef_state;
char utmd;
char osi_tp;
B  char certificate_gen;
char os[24];
char bit_mode[8];
char cluster_appl;
```

The fields in the data structure have the following meanings:

appliname
: The name of the application specified in the KDCDEF generation in MAX APPLINAME.

utm_version
: The UTM version used, including the update information, for example V06.4A00.

applimode
: Specifies if the UTM application is a UTM-S or UTM-F application.

  ´S´ The application is generated as a UTM-S application (Secure).

  ´F´ The application is generated as a UTM-F application (Fast).

system_type
: The operating system of the computer on which the application runs.

  ´B´ BS2000 systems

  ´X´ Unix and Linux systems

  ´N´ Windows systems

hostname
: The name of the computer on which the application runs.

destadm
: Contains the receiver to which UTM sends the results of KDCADM administration calls that were processed asynchronously (KDCADM asynchronous transaction codes). *destadm* may contain the following:

  – the name of an LTERM partner or
  – the transaction code of an asynchronous program unit.

  If *destadm* contains blanks, then no receiver is defined. The results of the KDCADM asynchronous transaction code are lost.

tacclasses
: Specifies if the application was generated with TAC classes, i.e. if TAC classes were created during the KDCDEF generation.

  ´Y´ The application was generated with TAC classes.

  ´N´ The application was generated without TAC classes.

pgwt Specifies whether program units containing blocking calls are allowed in the application (for example the KDCS call PGWT).

´Y´ Blocking calls are allowed, i.e. there is at least one transaction code or one TAC class with the property *pgwt*=´Y´ (see *kc_tac_str.pgwt* on page 555 and *kc_tacclass_str.pgwt* on page 560).

´N´ Blocking calls are not allowed, i.e. the application contains neither transaction codes nor TAC classes for which *pgwt*=´Y´.

B kdcload

B This field always contains ´N´.

B This field refers to functionality of UTM which is no longer supported.

load_module_gen
Specifies if the application was generated with load modules (BS2000 systems) or shared objects/DLLs (Unix, Linux and Windows systems), i.e. if at least one LOAD-MODULE statement or SHARED-OBJECT statement was specified for the KDCDEF generation.

´Y´ The application was generated with LOAD-MODULE or SHARED-OBJECT statements.

´N´ The application was not generated with LOAD-MODULE or SHARED-OBJECT statements.

prog_change_running
Specifies if UTM is currently executing a program change for the application.

´Y´ A program change is currently being executed.

´N´ No program change is currently being executed.

inverse_kdcdef_state
Specifies whether an inverse KDCDEF is currently running, i.e. if a KC_CREATE_STATEMENTS call is being processed.

´N´ No inverse KDCDEF is currently running.

´A´ An inverse KDCDEF run is being prepared. It will be started asynchronously as soon as all transactions that change configuration data have terminated. Administration calls that change configuration data will be rejected.

´Y´ An inverse KDCDEF is currently running.

utmd Specifies if the application is generated for distributed processing using a higher level communication protocol (LU6.1 or OSI TP).

´Y´ The application was generated for distributed processing.

´N´ The application was not generated for distributed processing.

osi_tp Specifies if the application is generated for distributed processing using OSI TP.

´Y´ The application was generated with statements for OSI TP.

´N´ The application was not generated with statements for OSI TP.

B B certificate_gen

Indicates whether users are generated with certificates.

B ´Y´ At least one user was generated with a certificate.

B ´N´ No users were generated with a certificate.

os Indicates the system platform of the computer, e.g. 'Windows Intel' or 'Solaris Sparc'.

bit_mode

Mode in which the operating system runs:

´32 bit´ 32-bit mode

´64 bit´ 64-bit mode

cluster_appl

Specifies whether the application belongs to a UTM cluster application.

´Y´ The application is a node application in a UTM cluster application.

´N The application is a standalone UTM application.

kc_tasks_par_str - Number of processes

The data structure *kc_tasks_par_str* is defined for the parameter type KC_TASKS_PAR. In the case of KC_GET_OBJECT, UTM returns all information on the processes of the application in *kc_tasks_par_str*:

- The maximum and current settings for the number of processes of the application.
- The maximum number of processes that may process asynchronous jobs at one time.
- The number of processes that may run at one time that contain program units with blocking calls (for example PGWT).
- The number of processed reserved for processing internal UTM jobs and dialog jobs, that do not belong to a dialog TAC class. This number is only returned if job processing is priority controlled in the application, i.e. if the TAC-PROPERTIES statement was set during KDCDEF generation.

mod[1]	Data structure kc_tasks_par_str
–	`char tasks[3];`
–	`char asyntasks[3];`
–	`char tasks_in_pgwt[3];`
x(A)	`char mod_max_tasks[3];`
x(A)	`char mod_max_asyntasks[3];`
x(A)	`char mod_max_tasks_in_pgwt[3];`
–	`char curr_max_asyntasks[3];`
–	`char curr_max_tasks_in_pgwt[3];`
–	`char curr_tasks[3];`
–	`char curr_asyntasks[3];`
–	`char curr_tasks_in_pgwt[3];`
x(A)	`char mod_free_dial_tasks[3];`
–	`char gen_system_tasks[3];`
–	`char curr_system_tasks[3];`

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 380f

The fields in the data structure have the following meanings:

tasks The control value generated for the maximum number of processes that may be used for the application.

The actual maximum number of processes that may process jobs of the application is determined at the start of the application and can be adjusted according to the actual demand during the application run (see *mod_max_tasks*). Neither the number of processes specified at the start nor the number set by the administration may exceed the value in *tasks*.

asyntasks
The control value generated for the maximum number of processes of the application that may be used for asynchronous processing at one time. The desired maximum number of processes for processing asynchronous jobs in the current application run can be set at the start of the application or by the administration (see the *mod_max_asyntasks* field). This number may not exceed the value of *asyntasks*.

tasks_in_pgwt
The control value generated for the maximum number of processes in which program units with blocking calls may run simultaneously (e.g. the KDCS call PGWT; Program Wait). The desired maximum number of processes for the current application run can be set at the start of the application or by the administration (see the *mod_max_tasks_in_pgwt* field). This number must not exceed the value of *tasks_in_pgwt*.

mod_max_tasks
Contains the current setting for the maximum total number of processes that may be used for the application at one time. *mod_max_tasks* contains the last setting of this number, which is either the number set at the start of the application or the number set by the administration (e.g. KC_MODIFY_OBJECT with KC_TASKS_PAR).

mod_max_tasks contains the set point for the current number of processes. The number of processes that are actually active currently and that can process the current jobs of the application is stored in the *curr_tasks* field. This may differ temporarily from the value in *mod_max_tasks* when a process is started or terminated, but only then.

Maximum value: *tasks*
Minimum value: ´1´

mod_max_asyntasks

Currently set limit value for the maximum number of processes that may be used for asynchronous processing. *mod_max_asyntasks* contains the last setting for the number of processes for asynchronous processing that was set either at start of the application or by the administration (e.g. KC_MODIFY_OBJECT with KC_TASKS_PAR).

The actual maximum number of processes that can be used at any one time for asynchronous processing (*curr_max_asyntasks*) can be lower than the value specified in *mod_max_asyntasks* because the actual number is limited by the number of currently running processes of the application (*curr_tasks*).

mod_max_asyntasks corresponds to a current upper limit.

Minimum value: ´0´
Maximum value: the number in *asyntasks*

mod_max_tasks_in_pgwt

Currently set limit value for the maximum number of processes in which program units with blocking calls may run simultaneously (Program Wait; e.g. the KDCS call PGWT). *mod_max_tasks_in_pgwt* contains the setting for number of processes that was set either at start of the application or by the administration (e.g. KC_MODIFY_OBJECT with KC_TASKS_PAR).

The actual maximum number of processes that process program units with blocking calls (*curr_max_tasks_in_pgwt*) at any one time can be lower than the value specified in *mod_max_tasks_in_pgwt* because the actual number must at least 1 less than the number of currently running processes of the application (*curr_tasks*).

mod_max_tasks_in_pgwt corresponds to a current upper limit.

Minimum value: ´0´
Maximum value: the number in *tasks_in_pgwt*

curr_max_asyntasks

The current maximum number of processes that may be used for asynchronous processing at one time. This number of processes is equal to whichever is lower of either the currently set maximum number of processes that can be used concurrently for asynchronous processing (*mod_max_asyntasks*) or the number of currently running processes of the application (*curr_tasks*). *curr_max_asyntasks* is changed dynamically by UTM when one of the two values *curr_tasks* or *mod_max_asyntasks* is changed. See also page 51.

curr_max_tasks_in_pgwt

Current setting for the maximum number of processes in which program units with blocking calls may run simultaneously (KDCS call PGWT). This number of processes is equal to whichever is lower of either the currently set maximum number of processes in which program units with blocking calls can run

concurrently (*mod_max_tasksk_in_pgwt*) or the number of currently running processes of the application (*curr_tasks*) minus one. *curr_max_asyntasks* is changed by UTM dynamically when one of the two values *curr_tasks* or *mod_max_tasks_in_pgwt* is changed. See also page 51.

curr_tasks
: Contains the number of processes of the application currently running. The value of *curr_tasks* usually corresponds to the value of *mod_max_tasks*. The value of *curr_tasks* can, however, be temporarily larger or smaller than *mod_max_tasks*. It is smaller if a process has terminated abnormally and has not been automatically restarted yet. It can be larger if the set point for the number of processes in *mod_max_tasks* was just recently lowered. *curr_tasks* contains the current value of the number of processes, *mod_max_tasks* contains the set point.

curr_asyntasks
: Contains the number of processes currently processing asynchronous jobs.

curr_tasks_in_pgwt
: Contains the number of processes currently processing program units with blocking calls (e.g. PGWT), i.e. the number of processes currently waiting in Program Wait.

mod_free_dial_tasks
: Only applies if the TAC-PRIORITIES statement was issued during the KDCDEF generation.

  UTM returns the current setting for the number of processes reserved for processing internal UTM tasks and jobs that are not assigned to a dialog TAC class in *mod_free_dial_tasks*. This portion of the total number of processes is consequently not available for processing jobs to dialog TAC classes.

  If the maximum number of application processes is reduced and this number is then smaller or equal to *mod_free_dial_tasks*, one process nevertheless processes jobs to dialog TAC classes.

  Minimum value: ´0´
  Maximum value: value in *tasks* -1

  If the application is generated without TAC-PRIORITIES, then UTM returns blanks in *mod_free_dial_tasks*.

gen_system_tasks
: Contains the maximum number of UTM system processes that can be started based on the current configuration.

curr_system_tasks
: Contains the number of currently running UTM system processes.

kc_timer_par_str - Timer settings

The data structure *kc_timer_par_str* is defined for the parameter type KC_TIMER_PAR. In the case of KC_GET_OBJECT, UTM returns the current settings for all timers of the UTM application in *kc_timer_par_str*.

The timers are set during the generation of the application and can be changed to adapt to the current situation during the application run using the operation code KC_MODIFY_OBJECT or with the help of the administration command KDCAPPL.

A change made to a timer only takes effect when the timer is reset for the first time after the change was made. The change does not affect timers already running. The changes are only in effect during the current application run.

mod[1]	Data structure kc_timer_par_str
x(GIR)	`char conrtime_min[5];`
x(GIR)	`char pgwttime_sec[5];`
x(GIR)	`char reswait_ta_sec[5];`
x(GIR)	`char reswait_pr_sec[5];`
x(GIR)	`char termwait_in_ta_sec[5];`
–	`char termwait_end_ta_sec[5];`
x(GIR)	`char logackwait_sec[5];`
x(GIR)	`char conctime1_sec[5];`
x(GIR)	`char conctime2_sec[5];`
x(GIR)	`char ptctime_sec[5];`
–	`char qtime1[5];`
–	`char qtime2[5];`

B

[1] The contents of the field can be modified with KC_MODIFY_OBJECT; see page 382f

The fields in the data structure have the following meanings:

conrtime_min

(**con**nection **r**equest **time**)
The time in minutes after which UTM is to attempt to re-establish a connection to a partner application, a client or a printer if the connection has been lost. *conrtime* is used for connections to:

- printers to which UTM automatically establishes a connection at the start of the application (*auto_connect*=´Y´ in *kc_pterm_str*, page 534) if this connection has not previously been shut down by administration functions.
- printers to which UTM establishes a connection as soon as the number of print jobs for this printer exceeds the generated control value (LTERM partner with *plev* > 0). UTM will only attempt to re-establish the connection if the number of print jobs that are still pending after the connection loss is greater than or equal to the control value.
  If *conrtime_min* ≠ 0, UTM will also attempt to establish the connection to the printer even if administration functions have previously been used to shut it down explicitly.
- TS applications (*ptype*=´APPLI´ or ´SOCKET´ in *kc_pterm_str*) to which UTM automatically establishes a connection at the start of the application (*auto_connect*=´Y´ in kc_pterm_str, page 534) if this connection has not previously been shut down by administration functions.
- OSI TP or LU6.1 partner applications to which UTM automatically establishes a connection when an application is started, if this connection has not previously been shut down by administration functions, or by UTM because a timer expired (*idletime*).
- B B B message routers (MUX) to which UTM automatically establishes a connection at the start of the application if it has not previously been shut down by administration functions.

If it is not possible to establish a connection to partners configured for automatic connection setup (at the start of an application or pursuant to a connection request issued using administration functions), or if such a connection is lost, then UTM will attempt to re-establish the connection, depending on the reason the connection was lost, in intervals of *conrtime_min*.

If *conrtime_min*=´0´, then UTM will not attempt to re-establish a logical connection.

Maximum value: ´32767´
Minimum value: ´0´

pgwttime_sec

The maximum amount of time in seconds that a program unit may wait for messages after a blocking function call. Blocking function calls are calls in which control is only returned to the program unit after the answer has been received (for example the KDCS call PGWT).

A process of the UTM application remains exclusively reserved for this program unit during this wait time.

Maximum value: ´32767´
Minimum value: ´60´

reswait_ta_sec

The maximum amount of time in seconds that a program unit may wait for a resource locked by another transaction: for example global secondary storage areas, user-specific long-term storage areas, terminal-specific long-term storage areas.

If the resource is not available after this time, then the application program will receive the appropriate return code KCRCCC.

The wait time specified in *reswait_ta_sec* is not significant if the lock is held by a multi-step transaction that is waiting for an input message (after a PEND KP or PGWT KP). In this case, all program units accessing the locked resource will immediately (without waiting for the time specified in *reswait_ta_sec*) receive the return code KCRCCC.

reswait_ta_sec=´0´ means that the program unit will not wait. A program unit run that attempts to access a locked resource will immediately receive the appropriate return code.

Maximum value: ´32767´
Minimum value: ´0´

reswait_pr_sec

The maximum amount of time in seconds that may be waited for a resource locked by another process. If this time is exceeded, then the application is terminated with an error message (see KC_MODIFY_OBJECT, *obj_type*=KC_TIMER_PAR as of page 382).

Maximum value: ´32767´
Minimum value: ´300´

term wait_in_ta_sec

The maximum amount of time in seconds that may pass between an output to a dialog partner and the reception of the dialog answer for multi-step transactions (i.e. in the PEND KP program). If the time *termwait_in_ta_sec* is exceeded, then the transaction is rolled back. The resources reserved by the transaction are released. The connection to the partner is closed.

Maximum value: ´32767´
Minimum value: ´60´

termwait_end_ta_sec

Does not contain a valid value. The time in seconds that UTM will wait for an input from the dialog partner after a transaction terminates or after signing on (KDCSIGN). This value is defined on a partner-specific basis as of openUTP V5.0. You will receive further information on this timer when you call KC_GET_OBJECT with *obj_type* KC_PTERM or KC_TPOOL (field *idletime* on page 539 and page 569).

B logackwait_sec

B The maximum amount of time in seconds that UTM is to wait for a logical print
B confirmation from the printer or for a transport confirmation for an asynchronous
B message sent to another application (created with the KDCS call FPUT).
B If the confirmation is not received after this time, for example because the printer is
B out of paper, then UTM closes the logical connection to the device.

B Minimum value: ´10´
B Maximum value: ´32767´

The following timers are only used for UTM applications with distributed processing via LU6.1 or OSI TP.

conctime1_sec

(**con**nection **c**ontrol **time**)
The time in seconds that the establishing of a connection to a session (LU6.1) or association (OSI TP) is to be monitored. If the session or association is not established within the specified time, then UTM closes the transport connection to the partner application. This prevents a transport connection from being disabled due to an unsuccessful attempt to establish a connection to a session or association. This can occur when a message needed to establish the connection becomes lost.

conctime1_sec=´0´ means that session setup is not monitored in the case of LU6.1 connections (UTM waits indefinitely). In the case of OSI TP connections, UTM waits for up to 60 seconds for an association to be set up.

Minimum value: ´0´
Maximum value: ´32767´

conctime2_sec

The maximum wait time in seconds that will be waited for the confirmation from the receiver when sending an asynchronous message. After the time in *conctime2_sec* runs out, UTM closes the transport connection. The asynchronous job is not lost, and it remains in the local message queue. Monitoring this time prevents a connection from not being used because a confirmation was lost and also prevents UTM from not being informed by the transport system of the loss of a connection.

conctime2_sec = ´0´ means that the connection will not be monitored.

Minimum value: 0
Maximum value: 32767

ptctime_sec

This parameter is only significant for distributed processing via LU6.1 connections. *ptctime_sec* specifies the maximum time in seconds that a job-receiving service in the PTC (prepare to commit, transaction status P) state will wait for confirmation from the job-submitting service. After this time is up, the connection to the job-submitter is closed, the transaction in the job-receiving service is rolled back and the service is terminated. This can eventually lead to a mismatch.

If KDCSHUT WARN or GRACE has already been issued for the application and the value of *ptc_time_sec* is not 0, then the waiting time is chosen independently of *ptc_time_sec* in such a way that the transaction is rolled back before the application is terminated in order to avoid abnormal termination of the application with with ENDPET if possible.

ptctime_sec = ´0´ means that there is no limit to the time that will be waited for confirmation.

Minimum value: 0
Maximum value: 32767

qtime1

Indicates the maximum time in seconds that a dialog service may wait for the receipt of messages for USER, TAC or temporary queues.

qtime2

Indicates the maximum time in seconds that an asynchronous service may wait for the receipt of messages for USER, TAC or temporary queues.

kc_utmd_par_str - Parameters for distributed processing

The data structure *kc_utmd_par_str* is defined for the parameter type KC_UTMD_PAR. In the case of KC_GET_OBJECT, UTM returns the basic settings for distributed processing via LU6.1 and OSI TP in *kc_utmd_par_str*.

Data structure kc_utmd_par_str
`char application_process_title[10][8];`
`char maxjr[3];`
`char rset;`

The fields in the data structure have the following meanings:

application_process_title
: Only of relevance for distributed processing via OSI TP.
application_process_title contains the application process title of the local application (see the openUTM manual “Generating Applications”).

  An application process title consists of at least two, but at most 10 components. Each individual component is a positive integer and is a maximum of 8 characters long.

  UTM returns one field element per component of the application process title, i.e. the number of field elements in *application_process_title* that contain data corresponds to the number of components generated. The rest of the field elements are set to binary zero.

  If no application process title was generated, then all field elements of *application_process_title* are set to binary zero.

maxjr (**maximum** number of **j**ob **r**eceivers)
: Specifies the maximum number of remote job-receiving services that may be addressed at one time within the local application.

  This value, in percent, corresponds to the total number of sessions and associations generated (=100%). The value must be between 0 and 200.

  A value greater than 100 means that openUTM APRO calls for addressing remote services are accepted, even if no session or association is (yet) free for this job at this time.

rset Specifies how rolling back a local transaction will affect the distributed transaction when distributed processing is utilized.

A local transaction can be rolled back by a RSET call from a program unit or by rolling back a database transaction that is involved in the local transaction. *rset* can contain one of the following values:

´G´ (GLOBAL)
After rolling back the local transaction the program unit must be terminated in such a manner that UTM rolls back the distributed transaction.

´L´ (LOCAL)
Rolling back a local transaction has no influence on the distributed transaction.

The distributed data can become inconsistent when some of the local transactions involved in a distributed transaction are rolled back and others continue as before. If *rset*=´L´, then global data consistency is not guaranteed by the system components involved. This task then becomes the responsibility of the application program units. They must decide in which situations the distributed transaction can be sensibly terminated and in which situations they must be rolled back.

12 Administration commands - KDCADM

This chapter describes the openUTM administration commands which call up the basic administration functions. These administration commands are transaction codes in the administration program KDCADM which are supplied together with openUTM. Before you can make use of these administration commands you must add entries for both KDCADM and the administration commands to the configuration file during the KDCDEF generation phase. The table below lists all the administration commands.

There are two versions of each administration command:

- a command to initiate processing interactively.
- a command for administration by means of message queuing (asynchronous processing).

There follows a description of the dialog-based administration commands. This description is arranged in ascending alphabetical order of command names. The associated commands for administration via message queuing have the same operands and the same input format. Command input differs only in terms of the actual command name entered.

With dialog-based administration procedures, openUTM returns the result of command processing to the job submitter (a user at a terminal, UPIC client, TS application or a partner application). In the case of asynchronous commands, all results are sent to a fixed receiver (DESTADM) in the form of asynchronous messages. The receiver can either be:

- an LTERM partner (Exception: UPIC LTERM partner are not permitted)
- an asynchronous TAC
- a TAC queue (TYPE=Q)

The receiver is defined during KDCDEF generation in MAX DESTADM= and can be modified via the administration program interface, see page 379. If no receiver is defined, the result is lost. If a TAC is defined but unable to receive the result, e.g. in the case of a disabled asynchronous TAC, UTM does not execute the administration command and writes message K076 to the system log file SYSLOG and to SYSOUT (on BS2000 systems) or *stderr* (on Unix, Linux and Windows systems).

List of administration commands

	Dialog commands	Asynchronous commands	Administration functions	Page
	KDCAPPL	KDCAPPLA	Change application parameter, timer and limit values	666
	KDCBNDL	KDCBNDLA	Replace Master LTERMs	680
	KDCDIAG	KDCDIAGA	Call diagnostic aids	681
	KDCHELP	KDCHELPA	Query syntax of administration commands	690
	KDCINF	KDCINFA	Query object properties, parameter values, statistics	691
	KDCLOG	KDCLOGA	Change user log file	742
	KDCLPAP	KDCLPAPA	Administer connections to partner applications	743
	KDCLSES	KDCLSESA	Establish and shut down logical connections for LU6.1 sessions	751
	KDCLTAC	KDCLTACA	Administer accesses to remote services (LTAC)	753
	KDCLTERM	KDCLTRMA	Change properties of LTERM partners	755
B	KDCMUX	KDCMUXA	Administer multiplex connections	758
	KDCPOOL	KDCPOOLA	Administer LTERM pools	762
	KDCPROG	KDCPROGA	Replace load modules or shared objects/DLLs	764
	KDCPTERM	KDCPTRMA	Change properties of clients/printers	769
B	KDCSEND	KDCSENDA	Send a message to one or more dialog terminals	774
	KDCSHUT	KDCSHUTA	Terminate an application program	775
	KDCSLOG	KDCSLOGA	Administer the system log file (SYSLOG)	779
	KDCSWTCH	KDCSWCHA	Change assignment of clients/printers to LTERM partners	785
	KDCTAC	KDCTACA	Disable/release transaction codes	789
	KDCTCL	KDCTCLA	Change number of processes for a TAC class	792
	KDCUSER	KDCUSERA	Change user properties	797

KDCADM commands

Command format

```
command-name␣operand1,operand2,...
```

There are position operands and keyword operands.

Position operands are entries without a keyword and an “=” sign and must appear in the specified sequence.
You can write keyword operands, e.g. PTERM=*ptermname* in any sequence. The operands must be separated by commas.

If an optional operand is not set, the value defined during generation or the value previously set by the administrator continues to apply.

After processing a command, openUTM returns an output which indicates the result. However, this does not mean in any case that the action was performed successfully.

You can use the information functions to establish whether or not openUTM was able to perform the action successfully, e.g. in the case of the command KDCINF.

KDCAPPL - Change properties and limit values for an operation

KDCAPPL allows you to perform the following actions:

- Change the timer settings and maximum values that you have generated in the KDCDEF control statement MAX.
- Define the number of processes (TASKS) that can be involved in an application simultaneously. If you wish to reduce the current number of processes, you should refer to the information provided on page 51.
- Define the maximum number of processes that are permitted to process asynchronous services or services with blocking function calls (e.g. the KDCS call PGWT) simultaneously. The maximum possible number of these processes depends on the total number of processes in the application and on the maximum number of processes defined in the KDCDEF statement MAX that are entitled to process services of this kind.
- Control cache memory paging.
- Switch the accounting and calculation phase of the UTM accounting procedure on and off
- Enable and disable data compression
- Establish a connection to all printers for which messages are present.
- Exchange the entire application program during live operation. This enables you, without terminating the application, to change program units and to take new program units that you have included dynamically in the configuration and add them to the application program.

  B B B In standalone UTM applications on BS2000 systems, load modules whose versions have previously been modified with KDCPROG are therefore also swapped in the Common Memory Pool.

- Switch over the system log files for the application (SYSOUT and SYSLST or stderr and stdout) during live operation. This avoids a hard disc bottleneck, since it allows log files to be evaluated and files which are no longer required to be deleted or archived.
- In addition, you can also enable or disable provision of data to the openSM2 software monitor for the application.

Period of validity of the changes

The changes made with KDCAPPL last for no longer than the duration of the current application run.

Exception:

A program replacement (PROGRAM operand) remains effective beyond the end of the current application run, i.e. when the application is next started, the most recently loaded version of the application program is reloaded. UTM even attempts to load the new application program if a previous attempt to initiate a program replacement (in the previous application run) was unsuccessful.

Effect in cluster operation

The effect on cluster applications is described in the sections devoted to the individual operands since some of the changes made with KDCAPPL apply locally to the node whereas others take effect globally in the cluster. Changes made locally in a node apply at the most for the duration of the current node application run.

Changes made globally in the cluster apply at the most for the duration of the current UTM cluster application run.

```
KDCAPPL␣   [ ,ACCOUNT={ ON | OFF } ]
           [  CACHE=%_utm_pages ]
           [ ,CALC={ ON | OFF } ]
           [ ,CONCTIME=con_control_time_sec ]
           [ ,CONRTIME=con_request_time_min ]
           [ ,DATA-COMPRESSION={ ON | OFF } ]
           [ ,MAXASYN=number_tasks ]
           [ ,MAX-CONN-USERS=number_users ]
           [ ,PGWTTIME=wait_time_sec ]
           [ ,PROGRAM={NEW | OLD | SAME} ]
           [ ,PTCTIME=wait_time_sec ]
           [ ,RESWAIT-PR=wait_time_sec ]
           [ ,RESWAIT-TA=wait_time_sec ]
           [ ,SPOOLOUT=ON ]
           [ ,SYSPROT=NEW ]
           [ ,TASKS=number_tasks ]
           [ ,TASKS-IN-PGWT=number_tasks ]
           [ ,TERMWAIT=wait_time_sec ]
           [ ,SM2={ ON | OFF } ]
```

For administration using message queuing you must specify KDCAPPLA.

ACCOUNT= enables and disables the UTM accounting phase.

In UTM cluster applications, the operand applies globally in the cluster.

B B B B On BS2000 systems enabling the accounting phase only becomes effective if record type UTMA is enabled on BS2000 Accounting. The command KDCAPPL ACC=ON is not rejected if record type UTMA is not enabled, since openUTM only detects this when writing an accounting record.

The value set in the KDCDEF control statement in ACCOUNT ACC= applies when the application is started.

For further details on UTM Accounting see also the openUTM manual “Using openUTM Applications”.

B B B B B B With KDCAPPL ACC=ON it is also possible to reenable the accounting phase in openUTM after BS2000 Accounting has failed, if BS2000 Accounting is available again.
In RAV (accounting procedure for computer centers) you can specify which tariffs are to be charged for specific periods of time when evaluating the accounting records.

CACHE=%_utm_pages
Defines the maximum percentage of pages in cache memory that can be stored on KDCFILE when bottlenecks develop.

In UTM cluster applications, the operand applies globally in the cluster.

CACHE allows you to adjust the percentage defined during KDCDEF generation in the MAX statement for the duration of the current application run. UTM swaps out at least 8 UTM pages in a paging operation, even a smaller value is calculated.

Minimum value: 0 (in this case, 8 pages are swapped out)
Maximum value: 100

If a value < 100 was specified at generation for *number* in the CACHESIZE operand of the MAX statement, rounding errors can occur at that output of *%_utm_pages* of a subsequent KDCAPPL command.

CALC= Enables the calculation phase of UTM Accounting.

In UTM cluster applications, the operand applies globally in the cluster.

ON Enables the calculation phase.

B B On BS2000 systems the calculation phase is enabled when the accounting record type UTMK is enabled on BS2000 Accounting.

OFF Disables the calculation phase again.

When the application is started, the value set in ACCOUNT ACC= in the KDCDEF control statement applies.

For further details on UTM Accounting see also the openUTM manual “Using openUTM Applications”.

CONCTIME=con_control_time_sec

(**Con**nection **C**ontrol **Time**)
Time taken in seconds to monitor the setup of a session (LU6.1) or an association (OSI TP). If the session or association is not established within the specified period of time, UTM shuts down the connection. This prevents a transport connection from remaining disabled due to a failure to establish a session or association.

In UTM cluster applications, the operand applies globally in the cluster.

CONCTIME=0 means

– session setup is not monitored in the case of LU6.1 session
– in the case of OSI TP connections, UTM waits up to 60 seconds for an association to be set up.

Default value: 0
Maximum value: 32767
Minimum value: 0

CONRTIME=con_request_time_min

(**con**nection **r**equest **time**)
Time in minutes for which UTM should continue attempting to re-establish a connection to a partner server or to a client or printer after a connection has been lost.

In UTM cluster applications, the operand applies globally in the cluster.

CONRTIME relates to connections to the following partners:

– Printers to which UTM automatically establish a connection when the application starts up. This assumes that the connection had not already been established by means of system administration functions.

– Printers to which UTM establishes a connection as soon as the number of print jobs for the current printer exceeds the generated control value (LTERM partner with *plev* > 0). UTM only attempts to re-establish a connection if the number of print jobs left after the lost connection is greater than or equal to the control value.
If in such a case CONRTIME ≠ 0, UTM attempts to establish a connection to the printer if the connection was previously shut down explicitly by means of system administration functions.

– TS applications which connect to the UTM application via LTERM partners (entered with *ptype*=´APPLI´ or ´SOCKET´) and to which UTM automatically establishes a connection when the application is started, unless this connection has already been cleared by the administrator.

– Partner applications to which UTM automatically establishes a connection at the start, unless such a connection has already been established by the administrator or shut down by UTM due to a timeout (*idletime*).

B
B
B

– Message routers (MUX), to which UTM should automatically establish a connection when the application is started if this connection has not already been cleared down previously by administration.

If no connection is established between partners configured with automatic connection setup (when an application is started or when an administration function requests a connection), UTM attempts to establish a new connection, or to re-establish the connection at the intervals specified in CONRTIME.

If CONRTIME=0, UTM makes no attempt to re-establish a connection.

Default value: 10
Maximum value: 32767
Minimum value: 0

DATA-COMPRESSION

Enables or disables data compression. Any modification applies beyond the end of an application.

The UTM pages saved per compression can be queried by means of WinAdmin/WebAdmin, program interface (see page 609), or KDCINF STAT command (see page 730).

In UTM cluster applications, the operand applies globally in the cluster.

ON Switches data compression on.
Data compression can be enabled using administration facilities only if it is permitted by means of generation (see openUTM manual “Generating Applications”, KDCDEF statement MAX DATA-COMPRESSION=).

OFF Switches data compression off.

MAXASYN=number_tasks

Specifies the maximum number of processes in the application that are allowed to accept jobs for asynchronous transaction codes at the same time (see KC_MODIFY_OBJECT, *obj_type*=KC_TASKS_PAR as of page 380, *mod_max_asyntasks* parameter).

In UTM cluster applications, the operand applies locally in the node.

MAX-CONN-USERS=number_users

Defines the maximum number of users who can have connections to a UTM application at the same time. This restriction enables you to prevent your application from becoming overloaded.

In UTM cluster applications, the operand applies locally in the node.

openUTM checks the number of active users when another user signs on, and rejects the connection attempt if the number of users defined in *number_users* are already signed on. This restriction does not apply to user IDs with administration privileges.

If, at the time of your KDCAPPL call, more than *number_users* users are working on the system, none of these users are forced to quit their application. However, no further connections will be permitted until the number of connected users falls to less than *number_users*.

If an application has been generated without user IDs, *number_users* restricts the number of dialog partners who can be connected to the application simultaneously. If a number is specified for *number_users* which is greater than the number of generated dialog LTERM partners, *number_users* has no effect. Dialog LTERM partners are all the LTERM partners generated with USAGE=D, LTERM partners of the LTERM pools and - with BS2000 - the LTERM partners UTM generates internally for multiplex connections.

number_users = 0 means that there is no restriction on the number of users or dialog partners working on the system.

Maximum value: 65000
Minimum value: 0 (no restriction).

PGWTTIME=wait_time_sec

Changes the maximum time in seconds defined during generation for which a program unit can wait for messages to arrive after a blocking function call; it also displays the currently set value for this wait time. The wait time is generated in the KDCDEF statement MAX with the operand PGWTTIME. Blocking function calls are calls where control is not returned until a response has been received by the program unit (e.g. the KDCS call PGWT).

In UTM cluster applications, the operand applies globally in the cluster.

During this wait time, a process in the UTM application is reserved exclusively for this program unit.

Maximum value: 32767
Minimum value: 60

PROGRAM= Replaces the entire application program during live operation (see the corresponding openUTM manual “Using openUTM Applications”).

Requirements for replacing a program using KDCAPPL PROGRAM=

– In the new application program, none of the earlier program units must be missing. UTM will terminate with errors (PEND ER) any jobs accepted for a transaction code for which no program unit exists after the program has been replaced.

– You should not initiate a program replacement until all previously entered UTM administration calls have been duly processed.
In particular, a program replacement should not be initiated until any program replacement already initiated has been fully processed, i.e. until the program replacement is complete for all processes in the application.

B B B – On BS2000 systems, a prerequisite for program replacement is that the application must have been generated with at least one LOAD-MODULE statement.

X/W X/W X/W X/W X/W X/W X/W X/W X/W – In order to be able to replace a UTM application program on Unix, Linux or Windows systems while the system is running, the various versions of the application program (including the currently loaded program) should be administered in the file generation directory PROG using the UTM tool KDCPROG. The file generation directory must have been created with KDCPROG and it must be in the *filebase* directory of your application. You can use PROG=OLD or NEW to switch to the previous or next file generation.
Program replacement is described in the openUTM manual “Using openUTM Applications on Unix, Linux and Windows Systems”

In UTM cluster applications, the operand applies globally in the cluster for all the node applications that are currently running.

Other KDCAPPL operands have no effect in conjunction with PROGRAM. KDCAPPL PROGRAM= is rejected if a program is being replaced at the time this call is submitted.

B B Only the values NEW and SAME are allowed on BS2000 systems. Both values have the same effect.

NEW The entire application program is to be replaced.

B *BS2000 systems*

B B KDCAPPL PROGRAM=NEW is not permitted if the application has been started in interactive mode.

B
B
B

In a UTM application on a BS2000 system the application program is unloaded in all processes and then re-loaded. When this is done, the current versions of the load modules are loaded.

B
B
B
B
B

Program units in common memory pools are also replaced at the same time. The version of load modules to be loaded during a program replacement must be declared in advance with KDCPROG. Before the program is replaced, several such load modules in the common memory pool can be earmarked for replacement by several KDCPROG calls.

B
B
B

In a standalone UTM application, it is possible to mark multiple load modules in the Common Memory Pool for replacement by means of multiple KDCPROG calls prior to replacement.

B
B
B

Termination of the application program followed by a reboot also causes all load modules generated with load mode ONCALL to be unloaded from the application.

B
B

This means that static parts of the application can also be replaced when the application is linked before.

B
B
B

In a UTM cluster application, application program replacement is initiated automatically as soon as the version of a load module in a Common Memory Pool is modified (via KDCPROG).

X/W

Unix, Linux and Windows systems

X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W

In a UTM application on Unix, Linux and Windows systems, openUTM loads the application program from the next highest file generation present in the file generation directory *filebase*/PROG (Unix and Linux systems) or *filebase*PROG (Windows systems) if the UTM tool KDCPROG is used to administer the different versions of the application program (including the one that is currently loaded) in the PROG file generation directory.
If you have not created a file generation directory, then KDCAPPL PROG=NEW reloads the application program (*utmwork)* that is located in the base directory *filebase*.

X/W
X/W
X/W
X/W
X/W
X/W

SAME

On Unix, Linux and Windows systems, openUTM loads the application program from the same file generation as is located in the file generation directory *filebase*/PROG (Unix and Linux systems) or *filebase*\PROG (Windows systems) if the different versions of the application program (including the currently loaded program) are administered using the UTM tool KDCPROG in the PROG file generation directory.

X/W
X/W
X/W

If you have not created a file generation directory, then KDCAPPL PROG=SAME reloads the application program (*utmwork*) that is located in the base directory *filebase*.

B

On BS2000 systems, SAME has the same effect as NEW.

X/W OLD openUTM loads the application program from the next lowest file generation present in the file generation directory *filebase*/PROG (Unix and Linux systems) or *filebase*PROG (Windows systems) if the UTM tool KDCPROG is used to administer the different versions of the application program (including the one that is currently loaded) in the PROG file generation directory.
This means, for example, that you can switch back to the original application program if errors are detected during operation with the new application program.
If you have not created a file generation directory, then KDCAPPL PROG=OLD reloads the application program (*utmwork)* that is located in the base directory *filebase*.

openUTM will not accept a new program replacement until the replacement has been completed for all the processes.

If errors occur in the first process when the new application program is started then openUTM issues the message K075 and loads the original application program again.

When the replacement of the application program has been terminated for all the processes, openUTM issues message K074.

X/W You can query the generation of the currently loaded application program, for example with KDCINF SYSP.

PTCTIME=wait_time_sec

Only for applications with distributed processing:
wait_time_sec is the maximum time in seconds which a local job-receiving service can wait for confirmation from the job-submitting service in PTC mode (prepare to commit, transaction status P). After this period has elapsed, the connection to the job submitter is shut down, the transaction is rolled back in the local job-receiving service and the service is terminated. This occasionally gives rise to a mismatch.

If KDCSHUT WARN or GRACE has already been issued for the application and the value of PTCTIME is not 0, then the waiting time is chosen independently of PTCTIME in such a way that the transaction is rolled back before the application is terminated in order to avoid abnormal termination of the application with ENDPET, if possible.

In UTM cluster applications, the operand applies globally in the cluster.

The value 0 indicates that the system can wait for an unrestricted length of time for a confirmation.

Maximum value: 32767
Minimum value: 0

RESWAIT-PR=wait_time_sec

The maximum time in seconds that one process can wait for resources that are being used by another process. If this period of time is exceeded, the application terminates with an error message.

In UTM cluster applications, the operand applies globally in the cluster .

If you specify RESWAIT-PR, please note that the value for *wait_time_sec* must be at least as long as the longest processing time (real time) required for the following cases:

– In the case of transport system applications that are not SOCKET applications (clients with PTYPE=APPLI), the resources can remain locked for the duration of one processing step, including the VORGANG exit at the start and/or end of the process.

– At the end of the process, the resources are locked until the VORGANG exit program stops running.

Maximum value: 32767
Minimum value: 300

RESWAIT-TA=wait_time_sec

Maximum time in seconds for which a program unit is to wait for a resource that is being used by another transaction: GSSBs, ULSs, TLSs.
If the resource does not become available after this time has elapsed, the an appropriate KCRCCC return code is sent to the application program.

In UTM cluster applications, the operand applies globally in the cluster.

The wait time specified in *wait_time_sec* is meaningless if the lock is effected by a multi-step transaction which is waiting for an input message after a PEND KP or a PGWT KP call. In this event, all program units with access to the locked resource immediately receive a KCRCCC return code (without wait time *wait_time_sec*).

RESWAIT-TA= 0 indicates that the process is not to wait. Any program unit run which attempts to access the locked resource immediately receives a KCRCCC return code. The real wait time depends on the precision with which the information exchange wait times are set in the operating system.

Default: 120
Maximum value: 32767
Minimum value: 0

SPOOLOUT=ON

Causes UTM to establish a connection to all printers for which messages exist at the time of the call and which are not yet connected to the application. This enables you to output all messages to those printers to which it is possible to establish a connection (e.g. before terminating the application).

In UTM cluster applications, the operand applies locally in the node.

SYSPROT=NEW

The log files of the application are switched over.

In UTM cluster applications, the operand applies globally in the cluster for all the node applications that are currently running.

The names of the new log files are formed as follows:

B *openUTM (BS2000 systems):*

B SYSOUT: *prefix*.O.*YYMMDD*.*HHMMSS*.*TSN*
B SYSLST: *prefix*.L.*YYMMDD*.*HHMMSS*.*TSN*

B Only SYSLST is switched over if the application is started interactively.

B prefix Prefix which you entered for the start parameter SYSPROT when
B the UTM application was started (see openUTM manual “Using
B openUTM Applications on BS2000 Systems”).

B Default value in standalone UTM applications:
B Name of the application as defined in MAX APPLINAME during
B KDCDEF generation.

B Default value in UTM cluster applications:
B Name of the application defined in MAX APPLINAME during the
B KDCDEF generation, followed by a dot and the computer name
B from the CLUSTER-NODE statement for the relevant node.

B YYMMDD
B Date at which the file was switched over.

B HHMMSS
B Time at which the file was switched over.

B TSN TSN of the task

X/W *openUTM on Unix, Linux and Windows systems:*

X/W stdout: *prefix*`.out.`*YY-MM-DD*`.`*HHMMSS*
X/W stderr: *prefix*`.err.`*YY-MM-DD*`.`*HHMMSS*

X/W prefix The prefix you specified for the start parameter SYSPROT when
X/W starting the UTM application (see openUTM manual “Using
X/W openUTM Applications on Unix, Linux and Windows Systems”).

X/W Default: utmp

X/W YY-MM-DD
X/W Date at which the file was switched over.

X/W HHMMSS
X/W Time at which the file was switched over.

TASKS=number_tasks

Specifies the current number of processes in the application. UTM switches processes on or off accordingly (see KC_MODIFY_OBJECT, *obj_type*=KC_TASKS_PAR as of page 380, *mod_max_tasks* parameter).

In UTM cluster applications, the operand applies locally in the node.

Maximum value:
The maximum value for TASKS defined during generation (KDCDEF control statement MAX...,TASKS=...)

Minimum value:
– If MAX TASKS-IN-PGWT=0: 1
– If MAX TASKS-IN-PGWT>0:
TASKS WAITING IN PGWT +1, but at least 2
(TASKS WAITING IN PGWT can be queried with KDCINF SYSP).

A certain amount of time is required for starting and terminating the UTM processes. After entering KDCAPPL TASKS= you should therefore first wait for the result of the call, then use KDCINF SYSPARM to check it before issuing any more KDCAPPL TASKS= calls. Failure to do so can lead to start errors.

TASKS-IN-PGWT=number_tasks

Defines the number of processes in the UTM application that are allowed to process program units in which blocking calls (e.g. the KDCS call PGWT) are permitted (see KC_MODIFY_OBJECT, *obj_type*=KC_TASKS_PAR as of page 380, *mod_max_tasks_in_pgwt* parameter).

In UTM cluster applications, the operand applies locally in the node.

The command is rejected if you enter TASKS-IN-PGWT=0, although MAX TASKS-IN-PGWT >0 was generated.

TERMWAIT=wait_time_sec

Maximum time in seconds allowed to elapse in a multi-step transaction (i.e. in the PEND KP program) between output to a dialog partner (terminal, UPIC client, TS application or job-submitting service in a partner application) and the ensuing dialog response. If this time exceeds *wait_time_sec* the transaction is rolled back. The connection to the dialog partner is shut down.

In UTM cluster applications, the operand applies globally in the cluster.

Maximum value: 32767
Minimum value: 60

SM2=

Switches the data supply to openSM2 on and off. It is only possible to supply data to openSM2 if the generation parameters permit openSM2 event logging to be switched on (KDCDEF statement MAX, operand SM2=ON or OFF). If SM2=NO is generated, the administrator will not be able to switch on the data supply to openSM2.

The following applies in UTM cluster applications:
The operand applies globally to the cluster. If a node application is started with a new generation then the value from the new generation applies for this node application as well as for all other newly starting node applications.

ON

openUTM supplies data to openSM2.

OFF

openUTM does not supply data to openSM2.

Output from KDCAPPL

The new and old values of the parameter are always displayed (with the exception of KDCAPPL PROG=).

```
                        NEW      OLD
TERMWAIT                ...      ...
RESWAIT-PR              ...      ...
RESWAIT-TA              ...      ...
CONRTIME                ...      ...
CURR TASKS              ...      ...
MAXASYN                 ...      ...
TASKS-IN-PGWT           ...      ...
CACHE                   ...      ...
ACCOUNT                 ...      ...
CALC                    ...      ...
SM2                     ...      ...
MAX-CONN-USERS          ...      ...
PGWTTIME                ...      ...
CONCTIME                ...      ...  1.
PTCTIME                 ...      ...  1.
X/W PROGRAM FGG         ...      ...  2.
```

1. The lines for PTCTIME and CONCTIME are only output via LU6.1 or OSI TP for an application with distributed processing.

X/W 2. The line for PROGRAM FGG is only output if in a UTM application on a Unix, Linux or Windows system the entire application program is to be replaced by means of KDCPROG. In such cases, the new generation number of the application program is output under NEW and the old generation number is output under OLD. Once all the processes in the program have been replaced, UTM issues message K074.

KDCBNDL - Replace Master LTERM

KDCBNDL allows you to exchange the Master LTERMs of two LTERM bundles (see openUTM manual “Generating Applications”). All the slave LTERMs and the associated PTERMs of the first LTERM bundle are assigned to the second master LTERM and vice versa.

This command is only permitted in standalone UTM applications.

Period of applicability of the change

The change remains effective after the application has terminated.

```
KDCBNDL    master-lterm1, master-lterm2
```

You must specify KDCBNDLA for administration via message queuing.

master-lterm1 Name of the first master LTERM.

master-lterm2 Name of the second master LTERM.

Output from KDCBNDL

The following messages are displayed at the administrator terminal:

- If the command was executed successfully:

  ```
  COMMAND ACCEPTED - 'master-lterm1' AND 'master-lterm2' SWITCHED
  ```

- If the command could not be executed:

  ```
  COMMAND REJECTED
  ```

- If an LTERM was specified for *master-lterm1* or *master-lterm2* that is not a master LTERM:

  ```
  COMMAND REJECTED - 'lterm' NO MASTER-LTERM
  ```

KDCDIAG - Switch diagnostic aids on and off

KDCDIAG allows you to switch UTM functions on and off which will support you during error diagnosis. The following functions can be called:

- Switch test mode on or off.
- Create a UTM dump during live operation.
- Cause openUTM to generate a dump when a specific event occurs.
- Switch UTM event monitor KDCMON on or off.
- Switch the BCAM trace function on or off. This function traces all connection-related activities in the application. The BCAM trace function can only be switched on for individual, explicitly specified communication partners.
- Switch the OSS trace function on or off. The OSS trace helps with the diagnosis of problems affecting OSI TP connections.
- Output debug information for the database connection.
- Switch over log files for the UTM application.

Effect in cluster operation

The effect on UTM cluster applications is described in the sections devoted to the individual operands since some of the changes made with KDCDIAG apply locally to the node whereas others take effect globally in the cluster. Changes made locally in a node apply at the most for the duration of the current application run. Changes made globally in the cluster apply at the most for the duration of the current cluster application run.

Period of validity of the changes

Every change remains effective for the duration of the application run or until it is reset.

```
KDCDIAG␣   [  DUMP=YES ]

           [ ,{ DUMP-MESSAGE | DUMP-MESSAGE{1|2|3} }=
                  { (MSG, msg-nr) | (SIGN, sign) | (RCCC, rccc) |
                    (RCDC, rcdc) | *NONE } ]

           [ ,INSERT1= (insert-nr, value, { EQ | NE }) ]

           [ ,INSERT2= (insert-nr, value, { EQ | NE }) ]

           [ ,INSERT3= (insert-nr, value, { EQ | NE }) ]

           [ ,KDCMON={ ON | OFF } ]

           [ ,TESTMODE={ ON | OFF } ]

           [ ,BTRACE= { ON | OFF } [ ,
                  { LTERM ={ ltermname | (ltermname_1,...,ltermname_10) } |
                    LPAP ={ lpapname | (lpapname_1,...,lpapname_10) } |
                    USER ={ username | (username_1,...,username_10) } |
B                   MUX  =( mux-name, proname, bcamappl )
                    } ]

           [ ,OTRACE= { ON | (SPI,INT,OSS,SERV,PROT) | OFF } ]

B          [ ,STXIT-LOG={ ON | OFF } ]

           [ ,XA-DEBUG={ YES | ALL | OFF} ]

           [ ,XA-DEBUG-OUT={ SYSOUT | FILE} ]
```

For administration using message queuing you must specify KDCDIAGA.

B DUMP=YES Requests a UTM dump during live operation. The UTM dump (taken from only one process in the application) is created with the reason "REASON=DIAGDP".

B In UTM cluster applications, the operand applies locally in the node.

DUMP-MESSAGE=[1...3]

Here, you specify an event that forces openUTM to create a UTM dump if it occurs. The command KDCDIAG DUMP-MESSAGE= is only evaluated when test mode is enabled (TESTMODE=ON).

In UTM cluster applications, the operand applies globally in the cluster .

The dump is created by the process in which the error occurs. The application is not terminated.

The parameters DUMP-MESSAGE1, DUMP-MESSAGE2 and DUMP-MESSAGE3 allow you to define up to three different events for which a message dump is to be generated if they occur. The specification DUMP-MESSAGE is synonymous with DUMP-MESSAGE1.

For each KDCDIAG demand, you can specify a maximum of one DUMP-MESSAGE[i] parameter.

You can specify the following events:

– the output of a message number with the format K*nnn* or P*nnn*
– the occurrence of a specific KDCS return code (CC or DC)
– the occurrence of a specific SIGNON status code

The dump ID is dependent on the event:

– In the case of K or P messages, it has the prefix "ME" followed by the message number, e.g. MEP012.

– In a primary KDCS return code, it has the prefix "CC-", followed by the return code, e.g. CC-71Z.

– In a secondary KDCS return code, it has the prefix "DC", followed by the return code, e.g. DCK303.

– In a SIGNON status code, it has the prefix "SG-", followed by the status, e.g. SG-U01.

MSG, msg-no

Specify UTM message number in the form K*nnn* or P*nnn* for *msg-no*. A dump is generated each time the specified message number occurs until such time as the message number is reset.

The message numbers K023, K043, K061 and K062 are exceptions in this regard, as only one dump is generated for each of them, after which the message dump is automatically disabled.

SIGN, sign

Specify a SIGNON status code (3 characters) for *sign*, e.g. U04, where KCRSIGN1 must have the value U, I, A or R. If this code occurs when a user signs on then the process in which the SIGNON status occurred generates a UTM dump with the ID SG-U04. This happens irrespective of whether a signon service has been generated in the application or not. The message dump for this event is then automatically deactivated.

RCCC, rccc
RCDC, rcdc

Specify a KDCS return code (KCRCCC, e.g. "40Z") for *rccc* and an incompatible KDCS return code (KCRCDC, e.g. "KD10") for *rcdc*. If this return code occurs on a KDCS call then the process in which the return code occurred generates a UTM dump with the ID CC-40Z or DCKD10. The message dump for this event is then automatically deactivated.

For all KDCS return codes >= 70Z and the associated incompatible KDCS return codes for which a PENDER dump is never generated (e.g. 70Z/K316), no dump is generated either.

*NONE Explicit deactivation of an event for a message dump.

INSERT1...INSERT3=(insert-no, value, {EQ | NE})

Here you can specify up to three inserts as additional conditions for the message *msg-no* in order to further restrict the generation of a dump. INSERTx is only evaluated if DUMP-MESSAGE[i] is also specified.

A message dump is only generated if all the criteria specified in INSERT1 ... INSERT3 are met.

You will find the sequence of the inserts of a message in the relevant, system-specific openUTM manual ”Messages, Debugging and Diagnostics”.

insert-no
Number of the insert to be checked, e.g. "2" for the second insert in a message.

value
Value against which the insert is to be checked. The following specifications are possible:
– nnn: numeric, value range $0...2^{31}-1$
– [C]’aaa’: alphanumeric, maximum length 32 bytes
– X’xxx’: hexadecimal, maximum length 32 bytes

EQ | NE
Specifies whether the system is to test for equality or inequality.
Default: EQ

KDCMON= Switches the UTM event monitor on or off.

In UTM cluster applications, the operand applies locally in the node.

ON Switches on the UTM event monitor.
The KCDMON event monitor is described in the corresponding openUTM manual “Using openUTM Applications”.

OFF Switches the UTM event monitor back off.

TESTMODE= Switches the test mode on and off.

In UTM cluster applications, the operand applies globally in the cluster.

ON Test mode is switched on. This means that additional internal UTM routines conduct plausibility checks and that internal TRACE information is recorded. Trace information is written in the KTA module and - with OSI TP applications - also in the XAPTP module. Trace mode should only be switched on in order to generate diagnostic documents.

OFF Switch test mode off.

Default: displays the current setting.

BTRACE= Switches the BCAM trace function of UTM on and off. The BCAM trace function of UTM traces all connection-specific activities occurring in a UTM application.
When the trace function (hereafter referred to as the BTRACE function) is switched on, every process in the application creates its own trace file in which it records all connection-specific events.
If the BTRACE function is switched off, the trace files are closed and can then be evaluated. For information about the contents and evaluation of trace files, please refer to the description in the openUTM manual ”Messages, Debugging and Diagnostics”.

The BTRACE function can be switched on using the start parameter BTRACE when the application is started.

You can switch on the BTRACE function for all connections in an application or on a partner-specific basis for connections to specified LTERM and LPAP partners.

ON The BTRACE function is usually switched on; it logs events relating to all connections to any given communication partner in the application (clients and partner applications in a distributed processing environment based on LU6.1).

In UTM cluster applications, the specification of BTRACE=ON without any other parameters applies globally in the cluster.

ON, LPAP=lpapname|(lpapname_1,...,lpapname_10)
In UTM cluster applications, the operand applies locally in the node.

or

ON, LTERM=ltermname|(ltermname_1,...,ltermname_10)
In UTM cluster applications, the operand applies locally in the node.

or

ON, USER=username|(username_1,...,username_10)
In UTM cluster applications, the operand applies globally in the cluster.

or

B B ON, MUX=(mux_name,proname,bcamappl)

In UTM cluster applications, the operand applies locally in the node.

The BTRACE function is switched on a partner-specific basis and all events relating to connections with specified partners or users or the MUX partners are logged.

The following specifications must be made:

- For *lpapname_...* the names of LPAP partners
- For *ltermname_...* the names of LTERM partners that are assigned to clients
- For *username_...* the names of users whose events are to be recorded or not recorded irrespective of the connection used. This is particularly useful when using TPOOLs.
- B B B For MUX the name and processor of the MUX partner and the transport system access point via which the MUX partner connects to the application.

The BTRACE function can only be switched on explicitly for connections with certain partner applications, clients or users if it is not already switched on for all connections in that application.

If you wish to switch on the BTRACE functions for a few partner applications and a few clients, call the KDCDIAG command repeatedly:

```
KDCDIAG BTRACE=ON,LPAP=...
KDCDIAG BTRACE=ON,LTERM=...
KDCDIAG BTRACE=ON, USER=...
```

B `KDCDIAG BTRACE=ON,MUX=...`

OFF The BTRACE function is switched off on all of the application's connections, even if it was originally switched on a partner-specific basis.

In UTM cluster applications, the specification of BTRACE=OFF without any other parameters applies globally in the cluster.

OFF, LPAP=lpapname|(lpapname_1,...,lpapname_10)

In UTM cluster applications, the operand applies locally in the node.

or

OFF, LTERM=ltermname|(ltermname_1,...,ltermname_10)

In UTM cluster applications, the operand applies locally in the node.

or

OFF, USER=username|(username_1,...,username_10)

In UTM cluster applications, the operand applies globally in the cluster.

or

B
B

OFF, MUX=(mux_name,proname,bcamappl)
In UTM cluster applications, the operand applies locally in the node.

This switches off the BTRACE function for connections to the partner applications specified in *lpapname_1,...,lpapname_10* or to the clients specified in *ltermname_1,...,ltermname_10* or to the users specified in *username_1,...,username_10* or the MUX partner.
The BTRACE function can only be switched off on a partner-specific basis if it had been switched on explicitly for connections to those partners (with BTRACE=ON, LPAP=... or LTERM=... or MUX=...).bzw. MUX=...

OTRACE= Switching the OSS trace function on and off.
The OSS trace is only required for the diagnosis of problems with OSI TP connections to the application. The OSS trace function can also be switched on or off when the application is started by appropriate entries in the start parameters [.UTM] START ... OTRACE=.

In UTM cluster applications, the operand applies globally in the cluster .

Trace records of the types SPI, INT, OSS, SERV and PROT are logged.

ON Switches on the OSS trace function for all types of record.
When the OSS trace function is switched on, every process in the application creates its own trace file.

(SPI, INT, OSS, SERV, PROT)
Switches the OSS trace function on. The trace records for the specified type are logged. The types can be entered in any sequence.

SPI
Logs the XAP-TP System Programming Interface.

INT
Logs the internal process in the XAP-TP module.

OSS
Logs the OSS calls.

SERV
Logs the internal OSS trace records of type =O_TR_SERV.

PROT
Logs the internal OSS trace records of type =O_TR_PROT.

OFF Switches off the OSS trace function; the trace files are closed and can be evaluated. For further details, please refer to the openUTM manual ”Messages, Debugging and Diagnostics” and the OSS manual.

B B B STXIT-LOG= Enable/disable extended STXIT logging in the event of STXIT handling problems. Several K099 messages are issued to SYSOUT when a STXIT event occurs.

B In UTM cluster applications, the operand applies locally in the node.

B ON Enables STXIT logging.

B OFF Disables STXIT logging.

XA-DEBUG= Specifies whether debug information for the XA database connection is to be output.

In UTM cluster applications, the operand applies locally in the node

YES XA-DEBUG is enabled, and calls to the XA interface are logged.

ALL Extended XA-DEBUG: specific data areas are output in addition to the calls to the XA interface.

OFF Disables XA-DEBUG.

XA-DEBUG-OUT=

Controls the output destinations for XA-DEBUG.

In UTM cluster applications, the operand applies locally in the node.

SYSOUT The log is written to SYSOUT/stderr, default.

FILE The log is written to a file.

If you specify only XA-DEBUG in the KDCDIAG command, without entering a value for XA-DEBUG-OUT then this may lead to a modification of the value which you specified in the start parameter when starting the UTM application (see openUTM manual “Using openUTM Applications”). Otherwise log entries are written to SYSOUT/stderr.

It only makes sense to change the two operands XA-DEBUG and XA-DEBUG-OUT in a UTM application in which a database connection has been generated over the XA interface.

Output from KDCDIAG

If KDCDIAG DUMP=YES then the message “DIAGNOSTIC DUMP CREATED” is issued. With the other operands, UTM displays the new and old settings for diagnostic aids on the administrator terminal:

```
STATUS          NEW                         OLD

TESTMODE        ON|OFF                      ON|OFF
KDCMON          ON|OFF                      ON|OFF
OSS-TRACE       ON|OFF                      ON|OFF
                SPI INT OSS SERV PROT       SPI INT OSS SERV PROT
BTRACE          ON S|ON A|OFF               ON S|ON A|OFF

LTERM/LPAP/USER BTRACE
                NEW                         OLD

ltermname       ON|OFF                      ON|OFF

lpapname        ON|OFF                      ON|OFF

username        ON|OFF                      ON|OFF

STXIT-LOG       ON|OFF                      ON|OFF

XA-DEBUG        YES|ALL|OFF                 YES|ALL|OFF

XA-DEBUG-OUT    SYSOUT|FILE                 SYSOUT|FILE
```

Explanation of the output

TESTMODE The line for TESTMODE is always displayed, regardless of whether or not the KDCDIAG call contains the TESTMODE operand.

KDCMON The line for KDCMON is always displayed, regardless of whether the KDCDIAG call contains the KDCMON operand or not.

BTRACE The line for BTRACE is always displayed. With BTRACE (ON) (switched on), the display also shows whether the trace function is switched on for all connections in the application (ON A, A=all) or just for connections to a few communication partners (ON S, S=select).

OSS-TRACE The line for OSS-TRACE is always displayed if the OTRACE operand was specified in the KDCDIAG call.

LTERM/LPAP/USER
Is only displayed if the BCAM trace function is/was explicitly switched on for connections to particular communication partners (LPAP, LTERM or MUX partners) or for users. The current and old BTRACE status is displayed for individual communication partners or users for whom the BTRACE function was switched on.

KDCHELP - Query the syntax of administration commands

KDCHELP provides you with information about the syntax of the administration commands.

```
KDCHELP␣   [command]
```

For administration using message queuing, you must enter KDCHELPA.

command

For *command*, enter the name of the administration command for which openUTM is to specify the syntax.

openUTM supplies the names of all KDCADM dialog commands together with a brief description of the functions of the individual commands if you enter:

`KDCHELP` (i.e. no entry made for *command*)
or
`KDCHELP KDCHELP` (i.e. KDCHELP is entered for *command*)
or
`KDCHELP XXX` (*XXX* = invalid name)

Valid entries for *command* are:

KDCAPPL	KDCLSES	KDCSLOG
KDCBNDL	KDCLTAC	KDCSWTCH
KDCDIAG	KDCLTERM	KDCTAC
KDCHELP	KDCPOOL	KDCTCL
KDCINF	KDCPROG	KDCUSER
KDCLOG	KDCPTERM	
KDCLPAP	KDCSHUT	

B B For UTM applications on BS2000 systems you can also enter the following names for *command*:

B B KDCMUX
KDCSEND

KDCINF - Request information on objects and application parameters

KDCINF allows you to query the names and properties of objects in an application as well as the application parameters generated and statistical information about the utilization level of the application. The parameter *type* allows you to define which information you require.

Effect in UTM cluster applications:

The information that is output always refers only to the local node application at which the job was executed.

Restricting the scope of information output

You can use a KDCINF call to query the properties of objects of a given type. The operands CONT, LIST and PRONAM define the scope of information that UTM is to output. You should define explicitly those objects about which you want UTM to provide information and define the scope of information for each object:

- In LIST you can explicitly define the names of the objects about which UTM is to provide information.
- By entering LIST=KDCNAMES you can restrict the output to a list of names of all objects of a given type. No other properties will then be displayed.
- LIST=KDCCON causes UTM to display only the properties of objects in a given type which are active at the current time, i.e. properties of clients, printers or partner systems to which there is a connection, or of users who are working on the system at the current time.
- CONT determines the object with which the list is to start. These lists are arranged alphabetically in order of object name. In CONT you enter a name. This can be any name - it does not have to be the name of an existing object. If the name you specify is the name of an object, the output list will start with that object. If there is no object with the name specified in CONT, the list will start with the object name that immediately follows the specified name in alphabetical order. No information is then provided about objects whose names come before the name specified in CONT when viewed in alphabetical order.
- With PRONAM you can restrict the output of object properties and names to objects located on a specific computer.

It is advisable in many cases to restrict KDCINF output, for example in large applications and for the output of information to a terminal. Full output of all information relating to one type of object is often so extensive that it can extend over many screen pages when output to the administrator’s terminal. It is then not possible to retain a clear overview. With large applications, generation of complete lists takes UTM a great deal of time. For this reason,

when dealing with larger applications, you should avoid requesting complete lists about objects of a given type, or lists of all objects and application parameters. In other words, avoid queries which take the following forms:

```
KDCINF ..,LIST=KDCALL,OUT=KDCPRINT or
KDCINF ..,LIST=KDCALL,OUT=KDCBOTH
```

Information output

The OUT operand allows you to define the location to which UTM is to output the requested information. You can also display information directly on the administrator terminal, send the information to a printer or transfer it directly to a program unit (asynchronous TAC) which will further process it.

For some objects, such as TAC, the output line of KDCINF does not provide sufficient space to display all the numeric values in their full length, for instance the value of the IN-Q field. When these values are too large to be displayed with KDCINF, they are truncated meaningfully and displayed in floating point presentation. In other words the leftmost digits are displayed, followed by an exponent e. The approximate actual value of the field is then obtained from the leading digits multiplied by 10 "to the power of" e.

Example:

When KDCINF TAC is output, four digits are available for the IN-Q field. If the number of messages which still needs to be processed by the TAC is greater than 9,999, the truncated presentation is used.

A value of 11,235 is displayed as 11e3, i.e. the actual value lies in the range between 11,000 and 11,999.

You will find examples of output and an explanation of the information that is output in the following description of operands (see page 703ff).

Special features of terminal output:
If the requested output does not fit on one screen, UTM displays a continuation command at the bottom of the screen (on the last line) which can be used to continue output from that position.

If you wish to page through the list using continuation commands, proceed as follows:

B
B

- on BS2000 systems:
  only overwrite one character in the specified command and press the <DUE> key.

X/W
X/W

- on Unix, Linux and Windows systems:
  enter the continuation command as displayed.

```
KDCINF␣    type
           [ ,LIST={ KDCNAMES | (name_1,...,name_10) | KDCALL | KDCCON } ]
           [ ,OUT={ KDCDISP | KDCPRINT | KDCBOTH | ltermname | tacname } ]
           [ ,CONT={ name | (name,proname) | (name,proname,bcamname) } ]
           [ ,PRONAM=proname]
           [ ,LPAP=lpapname ]                    (only for type = LSES)
           [ ,OSI-LPAP=osilpapname ]             (only for type = OSI-ASSOCIATIONS)
B          [ ,OPTION=MONITORING ]                (only for type = MUX)
```

For administration using message queuing, you must enter KDCINFA.

type

Type of objects or application parameters about which UTM is to provide information. For *type* you can enter the following values:

ALL	PROG	TAC
KSET	**PT**ERM	TACCLASS
LTERM	**STAT**ISTICS	**T**AC-**P**ROG
POOL	**S**YS**LOG**	**U**SER
PAGEPOOL	**SYSP**ARM	

To query information about objects in a distributed processing environment you can enter the following values for *type*:

CON	**OSI-ASS**OCIATIONS
LPAP	OSI-CON
LSES	OSI-LPAP
LTAC	

B
B

In UTM applications on BS2000 systems you can also enter the following for *type*:

B

LOAD-**MOD**ULE MUX

X/W
X/W

In UTM applications on Unix, Linux or Windows systems you can also enter the following for *type*:

X/W

SHARED-OBJECT

These entries for *type* have the following meanings:

ALL

Calls for all information about all objects, statistics and application parameters.

The result of KDCINF ALL is always output to the standard system printer (the default printer in the operating system). Control of output using the OUT operand is **not** possible: entries for OUT are ignored.
For the LIST operand, only the operand values KDCNAMES (default) and KDCALL are considered.

In the combination ALL and LIST=KDCNAMES, the names of all objects are output but no application parameters and no statistical information are displayed.

In the combination ALL and LIST=KDCALL, all application parameters, statistical information and the properties of all objects are displayed.

B
B

In UTM applications on BS2000 systems , no information is displayed about load modules in response to KDCINF ALL, LIST=KDCALL.

In KDCINF ALL, data relating to the CONT, OUT and PRONAM operands has no effect.

KSET Informs you about the key sets in the application. You can output information about a specific key set (use of LIST=kset_name) at an administrator terminal. If you wish to obtain information about several or all key sets, this information is always output to the standard system printer. Data relating to the OUT operand has no effect.

Exception

If a value greater than 255 was entered for the KEYVALUE operand in the MAX statement during KDCDEF generation, you **cannot** query any information about key sets with the KDCINF call (*type* = KSET). In this case, the KDCINF command is rejected and the message ”KEYVALUE > 255 NOT SUPPORTED“ is displayed.
However, you can create your own administration program for querying this information with the help of the administration program interface (KC_GET_OBJECT call with object type KC_KSET).

B LOAD-MODULE

B Informs you about load modules. The scope of output can be controlled
B using the CONT and LIST operands. For LIST, you are only allowed to enter
B KDCNAMES or an individual load module name. With LIST=KDCNAMES a
B list of all load module names is issued.
B You are provided with information about a specific load module if you enter
B the name of that load module in LIST.

B When you enter the name of a load module in LIST, the entry is interpreted
B as a program name in CONT. This entry in CONT determines the program
B unit name with which the list of program units in the load module should
B begin.

LTERM Informs you about LTERM partners, i.e. about the logical names and properties of clients and printers. The scope of this output can be controlled using operands CONT and LIST.

If an LTERM partner is assigned to a printer pool (several printers, i.e. PTERMs), then you can display the list of printers assigned to the LTERM (PTERMs) with the following command:

```
KDCINF LTERM, LIST=ltermname
```

If the specified LTERM is the primary LTERM of an LTERM group, the primary and group LTERMs of the LTERM group are output (see openUTM manual “Generating Applications”). The sequence is as follows:

– The first line contains the primary LTERM.
– The subsequent lines contain the group LTERMs.

If the specified LTERM is both the master LTERM of an LTERM bundle and the primary LTERM of an LTERM group, all master, primary, slave and group LTERMs are output. The sequence is as follows:
– The first line contains the master/primary LTERM.
– This is followed by all the group LTERMs.
– The subsequent lines contain the all the slave LTERMs. The first slave LTERM listed is the one to which the messages are delivered.

The call KDCINF LTERM, LIST=*master-lterm* outputs the master and slave LTERMs of an LTERM bundle. Output is the same as with the KDCINF LTERM, LIST=KDCALL call:
– The first line contains the master LTERM.
– The subsequent lines contain the slave LTERMs.

B MUX Informs you about the properties and current status of multiplex connections.
If MUX is entered together with the operand OPTION=MONITORING, UTM also supplies event values for the multiplex connections.
However, OPTION=MONITORING has no effect if you enter LIST=KDCNAMES.

PAGEPOOL
Informs you about the current utilization of the page pool.

Only the OUT operand is valid in conjunction with PAGEPOOL. Specifications for the LIST, CONT, and PRONAM operands are ignored by openUTM.

POOL Informs you about LTERM pools. The scope of output can be controlled with the operands CONT and LIST.

PROG This is only permitted if the application was generated using load modules/shared objects (the LOAD-MODULE statement (BS2000 systems) and the SHARED-OBJECT statement (Unix, Linux and Windows systems). openUTM informs you about the program units in the application.
In each program unit, the name of the relevant load module/shared object/DLL is displayed together with its load mode and a statement about how readily it can be replaced.
The scope of output can be controlled using the operands CONT and LIST.

PTERM Informs you about the physical properties of clients and printers. The scope of output can be controlled using the operands CONT, LIST and PRONAM.

X/W SHARED-OBJECT
This is only permitted if the application is generated using SHARED-OBJECT statements.
openUTM informs you about shared objects/DLLs.
The scope of output can be controlled using the operands CONT and LIST.

X/W
X/W
X/W

With LIST, the only entries permitted are KDCNAMES or an individual shared object name/DLL name. When LIST=KDCNAMES is entered, a list of all share object names or DLL names is issued.

STATISTICS

Displays general statistical information.
Together with STATISTICS, only the operand OUT has any effect. Entries for the operands LIST, CONT and PRONAM are ignored by openUTM.

Some statistical data is written to the system log file SYSLOG once an hour via the K081 message and subsequently reset to 0. For this to be possible, the application must have been generated with MAX STATISTICS-MSG=FULL-HOUR .The statistical values supplied by openUTM and their period of validity are described as of page 726.

SYSLOG Informs you about the SYSLOG file for the UTM application. Together with SYSLOG only the OUT operand has any effect. Entries for the operands LIST, CONT and PRONAM are ignored by openUTM.
KDCINF SYSLOG acts like KDCSLOG INFO (see page 779).

SYSPARM

Informs you about application parameters (system parameters) and timer settings which were defined during generation in the MAX statement and which can be changed using the administration functions. Using SYSPARM you can, for example, check parameter values which were changed using KDCAPPL.
Together with SYSPARM, only the operand OUT has any effect. Entries for the operands LIST, CONT and PRONAM are ignored by openUTM.

TAC Informs you about transaction codes or TAC queues in the application. The scope of output can be controlled with the help of operands CONT and LIST.

TAC-PROG

This is only permitted if the application was generated using load modules/shared objects (the LOAD-MODULE statement (BS2000 systems) or the SHARED-OBJECT statement (Unix, Linux and Windows systems) respectively.
openUTM informs you about which program units are assigned to the transaction codes and the load modules/shared objects/DLLs to which the program units are assigned. The transaction codes are specified in the LIST operand.

TACCLASS

Informs you about TAC classes in the application.
openUTM displays how many messages in each TAC class are waiting to be processed, how long the average wait time is for each TAC class and

whether priority control is generated for a TAC class. If priority control is not generated for a TAC class, i.e. the TAC-PROPERTIES statement was not issued during KDCDEF generation, openUTM displays how many processes are assigned to each TAC class.

USER Informs you about the user IDs in the application.
openUTM informs you about security violations for the user ID, CPU time used since the user signed on, and the LTERM partner used to sign on the user ID.

The scope of output can be controlled using the operands CONT and LIST.

The following values are only useful for applications that use distributed processing:

CON Only for distributed processing using the LU6.1 log.
openUTM informs you about connections that were created with KC_CREATE_OBJECT for the object type KC_CON or generated using the KDCDEF control statement CON. openUTM displays names, generated properties, current status and statistical values relating to the level of capacity utilization for the connection. The scope of output can be controlled using the operands CONT, LIST and PRONAM.

LPAP Only for distributed processing using the LU6.1 log.
openUTM informs you about the names and properties of the LPAP partners. Depending on the entries in LIST, open UTM will either display the names only, or these names together with the properties of the LPAP partners. The scope of output can be controlled using the operands CONT and LIST.

You can issue the call KDCINF LPAP, LIST=*master-lpap* to output the master and slave LPAPs of an LU6.1-LPAP bundle (see openUTM manual “Generating Applications”). The output has exactly the same form as for the call KDCINF LPAP, LIST=KDCALL:
– The first line contains the master LPAP.
– The following lines contain the slave LPAPs.

LSES Only for distributed processing using the LU6.1 log.
openUTM informs you about local sessions that were created with KC_CREATE_OBJECT for the object type KC_LSES or generated using the KDCDEF control statement LSES. If you specify LSES together with the operand LPAP=*lpapname* (KDCINF LSES,LPAP=*lpapname*), openUTM restricts output to information about sessions generated for the LPAP partner specified in *lpapname*. The scope of output can also be controlled with the aid of the operands CONT and LIST.

LTAC Informs you about the names and properties assigned to remote service programs within local applications (LTAC properties)
The scope of output can be controlled with the aid of the operands CONT and LIST.

OSI-CON Only for distributed processing using the OSI TP log.
openUTM informs you about names generated with the KDCDEF control statement OSI-CON for logical connections to partner applications.
Depending on the entries in LIST, the properties generated in OSI-CON for related connections are displayed.
The scope of output can be restricted with the help of the operands CONT and LIST.

OSI-LPAP Only for distributed processing using the OSI TP log.
openUTM informs you about OSI-LPAP partners that were generated for the OSI TP partner applications in the local application. Depending on the specifications made for the LIST operand, openUTM will either display the names only, or these names together with the logical properties of their partner applications.
The scope of output can be controlled with the aid of the operands CONT and LIST.

The KDCINF OSI-LPAP, LIST=*master-lpap-name* call outputs the master and slave LPAPs of an OSI-LPAP bundle (see openUTM manual “Generating Applications”). The output is the same as for the call KDCINF OSI-LPAP, LIST=KDCALL:
– The first line contains the master LPAP.
– The subsequent lines contain the slave LPAPs.

OSI-ASSOCIATIONS
Only with distributed processing using the OSI TP log.
openUTM informs you about OSI TP associations. Information about the job submitter assigning an association together with statistical information.

KDCINF...,L=KDCNAMES
outputs the name of generated OSI associations.

KDCINF...,L=KDCALL,OSI-LPAP=osilpapname
only outputs the currently associated OSI associations, sorted by the association ID assigned by XAPTP.
The operand OSI-LPAP=*osilpapname* is mandatory!

KDCINF...,L=(*name_1...name_10)*,OSI-LPAP=osilpapname
In this case, the association ID assigned by XAPTP must be entered for *name_n* rather than the OSI association name generated.
The operand OSI-LPAP=*osilpapname* is mandatory!

The scope of output can be controlled with the help of operands CONT and LIST.

openUTM restricts itself to information about OSI associations that have been set up in connection with the specified OSI-LPAP partners.

The following operands control output

LPAP=lpapname

is only permitted for *type* = LSES:
This operand restricts output of session properties to sessions that were generated for a partner application specified in *lpapname*.

B B B OPTION=MONITORING

is only permitted for *type* = MUX and only works where LIST ≠ KDCNAMES.
openUTM informs you about event values in multiplex connections.

B B With KDCINF ALL, these event values are not issued at the same time as the other values.

CONT= Continue/start the output list at a specific point. List output occurs in alphabetical order of object names. CONT=*name* causes the output list to start with the object *name* and to contain only objects whose name occurs following the one specified in *name* in alphabetical order.

B B B With UTM applications on BS2000 systems, the operand CONT, when specified in conjunction with LIST=*name*, only takes effect if the name of a program unit is entered for *type* LOAD-MODULE and for *name*.

X/W X/W X/W In UTM applications running on Unix, Linux or Windows systems the operand CONT has no effect if specified together with LIST=(*name_1*,..., *name_10*).

name The list starts with the object *name*. For *name,* specify the name of an object in the application. You can enter any one of the following names:

- for *type* = KSET: KSET name of a key set.
- for *type* = LTERM: logical name of a client/printer (name of an LTERM partner)
- for *type* =PTERM: (PTERM-)name of a client or printer
- for *type* =POOL: the LTERM prefix defined for an LTERM pool
- for *type* = PROG: name of a program unit
- for *type* =TAC: TAC name of a local transaction code/queue
- for *type* = USER: user ID (USER name)
- for *type* = CON/OSI-CON: a logical connection name generated in a CON or OSI-CON statement
- for *type* = LPAP/OSI-LPAP: name of an LPAP or OSI-LPAP partner
- for *type* = LTAC: local TAC name of a remote service program

– for *type*=OSI-ASSOCIATION: association ID, assigned to the association when establishing an OSI TP connection

B B In UTM applications on BS2000 systems you can also enter the following names:

B B
– for *type* = LOAD-MODULE: name of a load module or a program unit
– for *type* = MUX: name of a multiplex connection

(name,proname)

The list should begin with the object *(name,proname)*. *proname* is the name of the processor on which the object *name* is located. There is no point entering *proname* unless *type* = PTERM / CON / MUX objects of the same name exist, as a result of which unique identification is only possible using the different processor names.

(name,proname,bcamappl)

The list should start with the object (*name*,*proname*,*bcamappl*). *bcamappl* is the name of the transport access point that is used by the object (*name*,*proname*) to connect to the application. It is only of use to specify *bcamappl* if objects with *type*=PTERM / CON / MUX exist with the same name and processor and unique identification is therefore only possible if the name of the transport access point is different.

Output starts with the object *(name,proname)* to which the local transport access point name indicated in *bcamappl* is assigned.

LIST= Controls the type and scope of information.

KDCNAMES

Outputs a list of names of all objects of the type specified in *type*. LIST=KDCNAMES has no effect on *type*=PAGEPOOL, STATISTICS, SYSLOG, SYSPARM, TACCLASS.

(name_1,..., name_10)

The properties of objects with the names *name_1*,..., *name_10* are displayed. You can specify a maximum of 10 names. Parentheses are not required if only one name is specified.

KDCCON Only appropriate for *type* = PTERM, USER, LSES and CON.

B Also for UTM applications on BS2000 systems and for *type*=MUX.

Only the properties of objects currently connected to the application are displayed in response to this command.

Exceptions with type=USER:
If the application is generated with SIGNON MULTI-SIGNON=NO, then the user IDs through which only OSI TP partners are signed on to start asynchronous services are not displayed.

If the application is generated with SIGNON MULTI-SIGNON=YES, then the following user IDs are not displayed:
– user IDs with RESTART=NO that are not signed on through a terminal
– user IDs through which only OSI TP partners are signed on that have selected the functional unit “COMMIT” or that intend to start an asynchronous service.

KDCALL The properties of all objects of the type specified in *type* are displayed.

Default: KDCNAMES

OSI-LPAP=osilpapname
Only permitted for *type* = OSI-ASSOCIATIONS. The operand restricts the output of information to the OSI associations that have been established for the specified OSI-LPAP partner.

The operand must be specified for:
KDCINF OSI-ASSOCIATION...,L=(*name_1...name_10)*
KDCINF OSI-ASSOCIATION...,L=KDCALL

OUT= Indicates where UTM is to output the information requested.

KDCDISP Output to the administrator terminal, i.e. the terminal at which KDCINF was entered.

KDCPRINT

B/X Output to the standard system printer (default printer setting in the system).

B B On BS2000 systems, output takes the form of an EAM file rather than a SYSLST.

X X X X X On Unix and Linux systems, the shell script `admlp` is used for output. admlp is located in `$UTMPATH/shsc` and calls the `lp` command. Users can modify `admlp` or create their own script with the name `admlp` and store it under a separate directory. This directory must then be included in the path variable `$PATH` (prior to `$UTMPATH/shsc`).

W W Windows systems do not as yet support output to a printer from the UTM application, i.e. no file is generated or printed out.

KDCBOTH
Output to the administrator terminal and (on BS2000, Unix and Linux systems) to the standard system printer.

B B On BS2000 systems, output takes the form of an EAM file rather than a SYSLST.

X X On Unix and Linux systems, the shell script `admlp` is used for output (see above).

ltermname Output to the printer with the logical name *ltermname*.

tacname Name of the transaction code to which UTM is to transfer the result of the information query. The transaction code must be assigned to a program unit which runs in an asynchronous service.

Default: KDCDISP

PRONAM=proname

Only effective for *type* = PTERM, CON and MUX.
openUTM only supplies information about the clients and partner applications running on or connected to the computer *proname*.

Default value for openUTM for systems: Blanks for local devices

Output from KDCINF (examples)

Output is listed by *type*. The display shows all properties (LIST≠ KDCNAMES).
When you enter KDCINF ALL,LIST=KDCALL, with the exception of information about load modules and shared objects, all the items listed in the following section are output in succession.

Calculation of mean values

The mean values displayed with KDCINF are calculated as an arithmetic mean for the first 32767 values. After this, the new value is weighted with 1/32767. The previous mean value is weighted with 32767/32768.

Slight inaccuracies caused by rounding may occur in the calculation of the mean values. This is particularly the case, if the new value differs considerably from the mean value.

type=CON

```
CON  PRONAM    LPAP  BCAMAPPL  STA     CONNECT    CTIME     LETTERS   CONB

con  proname   lpap applname  ON| OFF  Y|N|W  A   minutes    number    number
```

Explanation of the output

CON — The name for the logical connections to the partner application *lpap* created with KC_CREATE_OBJECT for the object type KC_CON or generated with the KDCDEF control statement CON.

PRONAM — Name of the computer on which the partner application runs.

LPAP — Logical name of the partner application for which the logical connection was generated.

BCAMAPPL — Name of the local UTM application (BCAMAPPL name) via which the connection to the partner application is established.

STA — Status of the partner application:

ON:
The partner application is not disabled. A connection can be established with it, or a connection already exists.
OFF:
The partner application is disabled. A connection cannot be established.

CONNECT — Several items of information are supplied here.

1st column:
The partner application is connected to the application at this time (Y) or not (N), or UTM is attempting to establish a connection (W = waiting for a connection).

2nd column:
openUTM will establish the connection to the partner application automatically (A) when the application starts, or UTM will not attempt to establish an automatic connection when the application starts (no output).

CTIME — Duration of connection time in minutes.

LETTERS — Number of messages that have been exchanged via the connection since the application started, i.e. the number sent or received by the local application.
Every time the application starts, the counter is reset to 0.

CONB — Indicates how often the connection has failed since the application started. The CONB counter is reset to 0 every time the application starts.

type=KSET

The output illustrated below is only produced if a value ≤ 255 is specified in the MAX statement for the KEYVALUE operand during KDCDEF generation, i.e. if the application does not permit key codes with a number > 255.

```
KSET:kset
```

	0	1	2	3	4	5	6	.	.	.	18	19
0		x			x							
20				x		x	x					
40			x								x	
60												
80	x											
100												
120 . . .												
240												

Explanation of the output

KSET Name of the key set

In the first line of the output, all key codes between 1 and 19 in the key set are displayed. Line two contains all key codes with numbers between 20 and 39 etc.

The last line displays key codes with numbers between 240 and 259.

The output illustrated here signifies that key set *kset* contains key codes 1, 4, 23, 25, 26, 42, 58 and 80.

type=LOAD-MODULE

```
LOAD-MODULE          lmodname
VERSION (PREVIOUS)   previous element version
VERSION (CURRENT)    current element version
LIBRARY              name of program library
LOAD MODE            STATIC| STARTUP|? ONCALL| POOL| POOL/STARTUP| POOL/ONCALL
CHANGEABLE           YES| NO
AUTOLINK             YES| NO

PROGRAM LIST
program1             program2
program3             program4
```

Explanation of the output

LOAD-MODULE
Name of the load module up to 32 characters in length.

VERSION (PREVIOUS)
Preceding version of the load module

VERSION (CURRENT)
Currently loaded version of the load module

LIBRARY Name for the program library of up to 54 characters loaded from the load module

LOAD MODE Load mode for the load module; the following modes are possible:

STATIC The load module is incorporated as a static element of the application program.

STARTUP The load module is loaded dynamically as an independent unit whenever the application starts.

ONCALL The load module is loaded dynamically as an independent unit when a program unit or VORGANG exit assigned to the load module(s) is called for the first time.

POOL The load module is loaded into the common memory pool whenever the application starts. The load module does not contain a private slice.

POOL/STARTUP
The public slice of the load module is loaded into the common memory pool when the application starts. The private slice belonging to the load module is then loaded into the local process memory.

POOL/ONCALL
: The public slice of the load module is loaded into the common memory pool when the application starts. The private slice belonging to the load module is loaded into the local process memory when the first program unit assigned to this load module is called.

CHANGEABLE
: Display showing whether or not the load module can be replaced during live operation.

AUTOLINK
: Indicates whether the load module was loaded with the BLS autolink function.

PROGRAM LIST
: List of names of all program units and data areas (AREAs) assigned to the load module. The list also contains the names of all deleted objects.

type=LPAP

```
LPAP    KSET      STATUS     OUT-Q     IDLETIME   MASTER     BUNDLE
lpap    kset      ON| OFF  Q   number    seconds    master      M |Y | N
```

Explanation of the output

LPAP — Logical name of the partner application in the local application (name of the LPAP partner)

KSET — Key set assigned to the partner application. The key set defines access rights to the partner application for the local application.

STA — Status of the partner application:

1st column:
ON
The partner application is not disabled. A connection can be established or a connection already exists.
OFF
The partner application is disabled. No connection can be established.

2nd column:
Q (QUIET)
No more dialog jobs will be accepted for the partner application.

OUT-Q — Number of messages in the message queue that still have to be sent to the partner application.

IDLETIME — Time until the disconnection of an unused connection (session) between the partner application and the local application.

MASTER — If the LPAP partner forms part of an LU6.1-LPAP bundle then the name of the master LU6.1-LPAP of the bundle is displayed

BUNDLE — Specifies whether the LPAP partner belongs to an LU6.1-LPAP bundle.

M
The LPAP partner is the master LU6.1 LPAP of an LPAP bundle.

Y
The LPAP partner is a slave LU6.1 LPAP of an LPAP bundle.

N
The LPAP partner does not belong to an LU6.1-LPAP bundle.

type=LSES

```
LSES    RSES    LPAP    CON      PRONAM    BCAMAPPL  AG/USER

lses    rses    lpap    con      proname   applname  user
```

Explanation of the output

LSES Name of the LU6.1 session in the local application

RSES Name of the session in the partner application

LPAP Logical name of the partner application for which the session is generated.

CON, PRONAM, BCAMAPPL
Uniquely identifies the logical connection which was established for the session.

con is the name created with KC_CREATE_OBJECT for the object type KC_CON or generated with the KDCDEF control statement CON for the logical connections to partner application *lpap*.

proname is the name of the computer on which the partner application *lpap* is running.

applname is the name of the local UTM application (BCAMAPPL name), via which the connection to the partner application was established.

AG/USER Name of the job submitter for whom the session is reserved. *user* indicates who started the job-submitting service.

If the job-submitting service is running in the local application then the user ID or LTERM partner which started the service is entered against *user*.

If the job-submitting service is running in the partner application (the local application is processing the job) or if asynchronous messages are transferred to the session, the local session name (LSES name) is issued for *user*, i.e. the outputs for LSES and AG/USER are identical.

type=LTAC

```
LTAC   LOCK    STATUS    RTAC   CODE    LPAP  ACCESSWAIT   REPLYWAIT USED   D

ltac   number  ON| OFF   rtac  I| P| T  lpap  seconds      seconds   number D
```

Explanation of the output

LTAC — Local TAC name for the service program in the partner application

LOCK — Lock code assigned to the remote service in the local application (access protection); a number between 1 and 4000.

STATUS — The transaction code LTAC is disabled (OFF) or not disabled (ON).

RTAC — Name of the transaction code/service program in the partner application

CODE — Indicates which code type is used internally by UTM for the RTAC name.

- I — Integer code type
- P — PRINTABLE-STRING code type
- T — T61 string code type

LPAP — Logical name of the partner application in the local application (name of the LPAP partner).

ACCESSWAIT — Length of time openUTM waits for a session to be occupied (can include the time to establish a connection) when the remote service program starts; time shown in seconds.
If the LTAC is an asynchronous TAC then a wait time ≠ 0 signifies that the job is always entered in the message queue for the partner application.
The time is defined for KDCDEF generation and can be adjusted by administration (e.g. with the KDCAPPL TAC).

REPLYWAIT — Maximum length of time which UTM waits for a response from the remote service. The time is defined for KDCDEF generation and can be adjusted by administration (e.g. KDCAPPL TAC).

USED — Number of jobs issued to this LTAC since the application started.
The counter is reset to 0 every time the application starts.

D — Specifies whether the LTAC has been deletetd via dynamic administration or not (no entry).

type=LTERM

```
LTERM  PTERM  USER  KSET LOCK  USAGE      STATUS   OUT-Q    INCNT    SECCNT   D
lterm  pterm  user  kset lock  D|O        ON| OFF  number   number   number   D
                               B|M|S|
                               P|G|A
```

Explanation of the output

LTERM Name of the LTERM partner; logical name of the assigned client/printer.

PTERM Name of the client or printer (PTERM name) to which this LTERM partner is assigned.

USER User ID of the user currently connected to the application through this LTERM partner. If there is currently no connection, then *user_curr* contains blanks.

- If a connection exists:
  If no user has as yet been signed on at a terminal, *user_curr* also contains blanks.
- In applications with MULTI-SIGNON=YES:
  If a genuine user ID with RESTART=YES is signed on to an LTERM partner of a client of the type UPIC/APPLI/SOCKET, *user_curr* contains that user ID, otherwise it contains the connection user (*user_gen*).
- In applications with MULTI-SIGNON=NO:
  If a genuine user ID is signed on to an LTERM partner of a client of the type UPIC/APPLI/SOCKET, *user_curr* contains that user ID, otherwise it contains the connection user (*user_gen*).

KSET Key set assigned to the LTERM partner (access rights).

LOCK Lock code assigned to the LTERM partner (access protection).

USAGE	Type of LTERM partner 1st value: D: A client is assigned to the LTERM partner or O: A printer is assigned to the LTERM partner 2nd value: B: A printer bundle is assigned to the LTERM partner. M: The LTERM is a master LTERM of an LTERM bundle. S: The LTERM is a slave LTERM of an LTERM bundle. 3rd value: P: The LTERM partner belongs to an LTERM pool G: The LTERM is the primary LTERM of an LTERM group. A: The LTERM is an alias LTERM of an LTERM group.
STATUS	The LTERM partner is disabled (OFF) or not disabled (ON).
OUT-Q	Number of messages that still have to be output for the LTERM partner.
INCNT	Number of messages entered via this LTERM partner; if a printer is connected to the LTERM partner, the number of print job confirmations is entered here. The INCNT counter is reset to 0 every time the application starts.
SECCNT	Number of security violations on this LTERM partner since the start of the application (e.g. unauthorized codes entered). The SECCNT counter is reset to 0 every time the application starts.
D	Indicates whether the LTERM partner was deleted by dynamic administration (D) or not (no entry).

type=MUX

```
MUX    PRONAM   BCAMAPPL  STATUS   CONNECT   MAXSES   ACTCON   MAXCON
mux1   proname  applname  ON       Y   A     number   number   number
                          OFF      N
                                   W
```

Explanation of the output

MUX	Name of the multiplex connection
PRONAM	Name of the processor on which the message router runs.
BCAMAPPL	Name of the local application (BCAMAPPL name) through which the multiplex connection is established.
STATUS	The multiplex connection is disabled (OFF) or not disabled (ON).
CONNECT	Here, several items of information are provided.
	1st value: The multiplex connection is connected to the application (Y) or the multiplex connection is not connected to the application (N) or openUTM is attempting to establish a connection to the multiplex connection (W = waiting for connection)
	2nd value: When the application starts, openUTM automatically tries to establish a connection to the multiplex connection (A) or not (no entry)
MAXSES	Number of terminals that can be connected to the application at the same time using this multiplex connection.
ACTCON	Number of terminals that are at present connected to the application using this multiplex connection.
MAXCON	Maximum number of terminals that were connected to the application at the same time using this MUX connection. The MAXCON counter is reset to 0 every time the application starts.

MUX,OPTION=MONITORING

MUX	PRONAM	BCAMAPPL	LETTERS	INCNT	WAIT	SHORT	RTRYO	RTRYI
mux1	proname	applname	number	number	number	number	number	number

Explanation of the output

MUX	Name of the multiplex connection.
PRONAM	Name of the processor on which the message router is running.
BCAMAPPL	Name of the local application (BCAMAPPL name) via which connection to the multiplex connection is established.
LETTERS	Number of the input and output messages for this multiplex connection since the application started. The counter is reset to 0 every time the application starts.
INCNT	Number of input messages received through this multiplex connection. The INCNT counter is reset to 0 every time the application starts.
WAIT	Number of requests since the application started that were passed from BCAM to the multiplex connection requiring the resending of a message which it was previously not possible to accept because of a BCAM shortage (WAIT FOR GO) from BCAM.
SHORT	Number of BCAM shortages for this multiplex connection since the start of the application.
RTRYO	Number of tries to send an output message again since the start of the application (retry out).
RTRYI	Number of tries to read an input message again since the start of the application (retry in).

type=OSI-ASSOCIATIONS

```
ASSOC-ID  OSI-LPAP  OSI-CON   CONTWIN  CON-STATE   CONTIME   REQ-CALLS IND-CALLS
assoc-id  osi-lpap  osi-con   Y| N     CONNECTED|  minutes   number    number
                                       WAIT-GO|
                                       STOP
```

Explanation of the output

ASSOC-ID ID assigned to the association when it was established. This is only unique while the association remains established. If the association is terminated, the ID is released and can be assigned to another established association.

OSI-LPAP Logical name of the partner application (name of the OSI-LPAP partner) for which the association was generated.

OSI-CON The OSI-CON name generated with the KDCDEF control statement for the logical connection to the partner application *osi-lpap*. If no connection is established, blanks are output at this point.

CONTWIN Indicates whether the local application for this association is the contention winner or the contention loser.

CON-STATE Indicates the status of the association.

CONNECTED
The association is established.

WAIT-GO The association is being established. It is waiting for a “GO” from OSS.

STOP The association is being established. The OSS call *a_assrs* has run up to “STOP”.

CONTIME Indicates the length of time in minutes for which the connection has existed.

REQ-CALLS Number of request/response presentation calls to OSS since the association was established.

IND-CALLS Number of indication/confirmation presentation calls to OSS since the association was established.

type=OSI-CON

```
OSI-CON   OSI-LPAP    T-SEL     N-SEL     ACC-PNT        ACTIVE

osi-con   osi-lpap    applname  proname   access-point   Y | N
```

Explanation of the output

OSI-CON — Name of the logical connection to the partner application *osi-lpap*.

OSI-LPAP — Logical name of the partner application in the local application (name of the OSI-LPAP partner) for which the connection was generated.

T-SEL — Application name of the partner application in the local system (transport selector).

N-SEL — Logical name of the computer on which the partner application is running (network selector).

ACC-PNT — Name of the local access point via which the connection *osi-con* is established.

ACTIVE

- Y — Connection *osi-con* can be used, i.e. messages for the specified OSI-LPAP partner are sent through and received by this connection.
- N — The *osi-con* connection cannot be used. It is reserved as a replacement connection.

type=OSI-LPAP

```
OSI-LPAP KSET   STA    Q    OUT-Q  IDLET   OSI-CON ASSOC  CONN   AUTOC  BU
osi-lpap kset   ON|OFF Q    number seconds osi-con number number number M|S
```

Explanation of the output

OSI-LPAP — Logical name of the partner application in the local application (name of the OSI-LPAP partner).

KSET — Key set assigned to the OSI-PAP partner. The key set defines access rights to the partner application for the local application.

STATUS — Status of the partner application:

ON
The OSI-LPAP partner is not disabled. A connection to the partner application can be established or a connection already exists.
OFF
The OSI-LPAP partner is disabled. A connection cannot be established to the partner application.

Q (Quiet): — No more dialog jobs will be accepted for the OSI-LPAP partner.

OUT-Q — Number of messages in the message queue which still have to be sent to the partner application.

IDLETIME — Time to monitor the idle state of sessions between the partner application and the local application.

OSI-CON — The OSI-CON name generated with the KDCDEF control statement for the logical connection to the partner application.

ASSOC — Maximum number of parallel connections (associations) to the OSI-LPAP partner that can exist at the same time. The number is defined during KDCDEF generation in the OSI-LPAP statement.

CONNECT — Number of connections to the OSI-LPAP partner existing at the present time or currently being set up.

AUTOCON — Number of connections to the OSI-LPAP partner which UTM should establish automatically when the application starts.

BU (bundle) — Specifies whether the OSI-LPAP partner belongs to an OSI-LPAP bundle.

M
The OSI-LPAP partner is the master LPAP of the OSI-LPAP bundle.

S
The OSI-LPAP partner is a slave LPAP of the OSI-LPAP bundle.

type=PAGEPOOL

```
PAGEPOOL INFORMATION
percent % PAGES FOR GSSB                  percent % PAGES FOR LSSB
percent % PAGES FOR TLS                   percent % PAGES FOR ULS
percent % PAGES FOR DIALOG SERVICES       percent % PAGES FOR TAC-CLASSES
percent % PAGES FOR FPUT-MANAGEMENT       percent % PAGES FOR ASYN MESSAGES
percent % PAGES FOR MSGTAC MESSAGES       percent % PAGES FOR LPUT
percent % PAGES FOR PHYS. MESSAGES        percent % PAGES FOR RESET MESSAGES
percent % PAGES FOR OSI TP LOG RECORDS    percent % OTHER PAGES
percent % FREE PAGES
```

Explanation of the output

PAGES FOR GSSB
: Number of pages, in percent, which are utilized for GSSBs.

PAGES FOR LSSB
: Number of pages, in percent, which are utilized for LSSBs.

PAGES FOR TLS
: Number of pages, in percent, which are utilized for TLS areas.

PAGES FOR ULS
: Number of pages, in percent, which are utilized for ULS areas.

PAGES FOR DIALOG SERVICES
: Number of pages, in percent, which are utilized for service contexts by users.

PAGES FOR TAC-CLASSES
: Number of pages, in percent, which are utilized for dialog input messages, and which are temporarily stored in TAC Class Queues.

PAGES FOR FPUT-MANAGEMENT
: Number of pages, in percent, which are utilized for managing asynchronous messages.

PAGES FOR ASYN MESSAGES
: Number of pages, in percent, which are utilized for asynchronous messages.

PAGES FOR MSGTAC MESSAGES
: Number of pages, in percent, which are utilized for MSGTAC messages.

PAGES FOR LPUT
: Number of pages, in percent, which are utilized for temporarily stored LPUT records.

PAGES FOR PHYS. MESSAGES
: Number of pages, in percent, which are utilized for output messages and which need to be temporarily stored because they can only be transferred to the transport system in sections owing to their length.

PAGES FOR RESET MESSAGES
: Number of pages, in percent, which are utilized for reset messages.

PAGES FOR OSI TP LOG RECORDS
: Number of pages, in percent, which are utilized for OSI TP log records.

OTHER PAGES
: Number of other utilized pages, in percent.

FREE PAGES
: Number of free pages, in percent.

In the case of UTM cluster applications, GSSB and ULS areas are stored in the global page pool of the UTM cluster application. As KDCINF PAGEPOOL only displays the utilization of the local page pool, the values for GSSB and ULS are always zero in UTM cluster applications.

type=POOL

```
POOL      PRONAM   BCAMAPPL  PTYPE  STATIONS  STA=ON   ACTCON  MAXCON   KSET  LOCK
ltprefix  proname  applname  ptype  number    number   number  number   kset  lock
```

Explanation of the output

POOL — LTERM prefix of the LTERM pool. The names of the LTERM partners assigned to the pool comprise *ltprefix* and a serial number between 1 and the maximum number of clients allowed to connect to the LTERM pool at the same time.

PRONAM — Only clients located on computer *proname* can establish connections to the application using the LTERM pool.

X/W X/W X/W — In the case of applications on Unix, Linux or Windows systems, blanks are entered for *proname* if the LTERM pool is defined for locally connected clients.

BCAMAPPL — Name of the local application (BCAMAPPL name) through which the connections to this LTERM pool were established (see KDCDEF statement TPOOL operand BCAMAPPL).

PTYPE — Physical type of client allowed to connect to the application through this LTERM pool.

PTYPE — Physical type of client allowed to connect to the application through this LTERM pool.

STATIONS — Maximum number of clients allowed to connect to the application at the same time using this LTERM pool.

STA=ON — Number of clients currently allowed in the LTERM pool.

ACTCON — Number of clients connected to the application at the present time through this pool.

MAXCON — Maximum number of clients connected to the application during the current application run using this LTERM pool.
The counter is reset to 0 every time the application starts.

KSET — Key set assigned to the LTERM pool, and therefore to all clients connected to the application by this LTERM pool (access rights).

LOCK — Lock code assigned to the LTERM pool (access protection).

type=PROG

For KDCINF PROG,L=KDCALL,CONT=*programname*

B *Output for UTM applications on BS2000 systems*

B	PROGRAM	LOAD-MODULE	L-MODE	CHN	D
B	program1	load module1	load mode	YES | NO	D
B	program2	load module2	load mode	YES | NO	

X/W *Output for UTM applications on Unix, Linux and Windows systems*

X/W	PROGRAM	SHARED-OBJECT	L-MODE	CHN	D
X/W	program1	shared object1	load mode	YES | NO	D
X/W	program2	shared object2	load mode	YES | NO	

Explanation of the output

PROGRAM Name of the program unit as specified during generation in the PROGRAM statement; up to 32 characters in length.

B LOAD-MODULE
B Name of the load module to which this program unit is assigned; up to
B 32 characters in length.

X/W SHARED-OBJECT
X/W Name of the shared object/DLL to which this program unit is assigned; up to
X/W 32 characters in length.

L-MODE Load mode of the load module/shared object/DLL to which this program unit is assigned. Key to terms:

STATIC
The load module/shared object/DLL is incorporated as a static element in the application program.

STARTUP
The load module/shared object/DLL is loaded dynamically as an independent unit whenever the application is started.

ONCALL
The load module/shared object/DLL is loaded as an independent unit whenever a program unit or VORGANG exit assigned to the load module/shared/DLL is called for the first time.

B POOL
B The load module is loaded into the common memory pool whenever the
B application starts. The load module does not contain a private slice.

B
B
B
B

POOL/STARTUP
The public slice of the load module is loaded into the common memory pool whenever the application starts. The private slice belonging to the load module is then loaded into the local process memory.

B
B
B
B
B

POOL/ONCALL
The public slice of the load module is loaded into the common memory pool whenever the application starts. The private slice belonging to the load module is loaded into the local process memory when the first program unit assigned to this load module is called.

CHANGEABLE
Displays whether or not the load module/shared object/DLL to which this program unit is assigned can be exchanged.

D Indicates whether the program was deleted from the configuration by means of system administration functions (D) or not (no entry).

type=PTERM

```
PTERM     PRONAM          LTERM      BCAMAPPL   PTYP
pterm     proname         lterm      applname   ptype

STA       CONNECT         CTIME      LETTERS    CONB       D
ON | OFF  Y|N|W  A|P   M  minutes    number     number     D
          T|E
```

Explanation of the output

PTERM — Name of the client or printer (PTERM name).

PRONAM — Name of the computer on which the client/printer is located.

X/W X/W — In UTM applications on Unix, Linux or Windows systems blanks are output for local clients/printers.

LTERM — Name of the LTERM partner (logical name) to which this client/printer is assigned.

BCAMAPPL — Name of the local UTM application (BCAMAPPL name) via which the connection to the client/printer is established.

PTYP — Type of client/printer (for the meaning of the output, see table on page 531 (BS2000 systems) or on page 534 (Unix, Linux and Windows systems)

STA — The client/printer is disabled (OFF) or not disabled (ON).

CONNECT Several items of information are provided here.

1st column

Y/N/W The client/printer is at present connected to the application (Y) or not (N), or UTM is now attempting to establish a connection (W = waiting for connection)

T/E Only output for terminals connected to a UTM application on a BS2000 system by a multiplex connection.
T: (timer) The session is in DISCONNECT-PENDING mode; the timer is running, waiting for confirmation that a connection is being established.
E: (expired) The session is in DISCONNECT-PENDING mode and the timer waiting for confirmation has run out before confirmation was received.

In both cases, the session can be released with KDCPTERM.

2nd column:

A automatic connection is established when the application starts or

P the client is connected to the application by an LTERM pool.
If neither of these properties applies, there is no output at this point.

3rd column (only for UTM applications on BS2000 systems):
The client is connected to the application by a multiplex connection (M) or not (no entry).

CTIME Duration of the existing connection in minutes.

LETTERS Number of input and output messages for the client or output messages to the printer since the application started.
The counter is reset to 0 every time the application starts.

CONB Number of connection failures between client/printer and application since the application started. The CONB counter is reset to 0 every time the application starts.

In UTM applications on BS2000 systems, the CONB counter also counts incrementally if a UPIC client first shuts down its connection to the UTM application and then establishes a new connection using the same PTERM name.

D Indicates whether the client/printer was deleted (D) or not (no entry) from the configuration by means of system administration functions.

X/W **type=SHARED-OBJECT**

X/W For KDCINF SHARED-OBJECT, L=*shared-object-name*, CONT=*programname*

```
X/W  SHARED-OBJECT       shared object name
X/W  VERSION (PREVIOUS)  old  version
X/W  VERSION (CURRENT)   new  version
X/W  LIBRARY             name of program directory
X/W  LOAD MODE           STATIC | STARTUP |? ONCALL ?
X/W  CHANGEABLE          YES | NO

X/W  PROGRAM LIST
X/W  program1            program2
X/W  program3            program4
```

X/W *Explanation of the output*

X/W SHARED-OBJECT
X/W Name of the shared object/DLL; up to 32 characters in length.

X/W VERSION (PREVIOUS)
X/W Previous version of the shared object/DLL.

X/W VERSION (CURRENT)
X/W Currently loaded version of the shared object/DLL.

X/W LIBRARY Name for the program library of up to 54 characters loaded from the shared
X/W object/DLL.

X/W LOAD MODE
X/W Load mode for the shared object/DLL. Key to the terms used:

X/W STATIC The shared object/DLL is incorporated as a static element in the application
X/W program.

X/W STARTUP The shared object/DLL is loaded dynamically as an independent unit
X/W whenever the application starts.

X/W ONCALL The shared object/DLL is loaded as an independent unit whenever a
X/W program unit or VORGANG exit assigned to the same shared object/DLL(s)
X/W is called for the first time.

X/W CHANGEABLE
X/W Displays whether or not the shared object/DLL can be replaced during run
X/W time.

X/W PROGRAM LIST
X/W List of the names of all program units and data areas (AREAs) assigned to
X/W the shared object/DLL.

type=STATISTICS

```
name   APPLINAME                        version   VERSION OF UTM
yy-mm-dd   DATE OF START APPLICATION       hh:mm:ss   TIME OF START APPLICATION
yy-mm-dd   CURRENT DATE                    hh:mm:ss   CURRENT TIME
  number   TERMINAL INPUT MESSAGES           number   TERMINAL OUTPUT MESSAGES
  number   CURRENT TASKS                     number   CONNECTED USERS
  number   OPEN DIALOG SERVICES              number   OPEN ASYN SERVICES
 percent % CURRENT LOAD                     percent % MAXIMUM LOAD
  number   DIALOG TAS PER SECOND            number   ASYN TAS PER SECOND
  number   DIALOG STEPS PER SECOND         percent % MAXIMUM POOL SIZE
 percent % ACTUAL POOL SIZE                 percent % AVERAGE POOL SIZE
 percent % CACHE HIT RATE                    number   NR CACHE SEARCHES
 percent % CACHE WAITS FOR BUFFER            number   NR CACHE REQUESTS
  number   UNPROCESSED ATACS                 number   UNPROCESSED PRINTS
  number   WAITING DPUTS                     number   ABNORMAL TERMINATED SERVS
  number   LOGFILE WRITES                    number   UTM-DEADLOCKS
  number   PERIODIC WRITES                   number   PAGES PER PERIODIC WRITE
 percent % WAITS FOR RESOURCES               number   NR RESOURCE REQUESTS
 percent % MAX WAITS FOR RESOURCES           number   NR RES REQUESTS FOR MAX
 percent % WAITS FOR SYSTEM RES              number   NR SYSTEM RES REQUESTS
 percent % MAX WAITS FOR SYSTEM RES          number   NR SYSTEM RES REQ FOR MAX
 percent % ACTUAL JR                        percent % MAXIMUM JR
  number   AVG COMPRESS PAGES SAVED
```

Explanation of the output

APPLINAME Name of the application that was defined during KDCDEF generation for MAX APPLINAME.

VERSION OF UTM
openUTM version used with correction status

DATE OF START APPLICATION
Day of the last cold start of the application (UTM-S application); day of the last start of the application (UTM-F application)

TIME OF START APPLICATION
Time of the last cold start of the application (UTM-S application); Time of the last start of the application (UTM-F application)

TERMINAL INPUT MESSAGES
Total number of messages received by the application from clients or partner applications during the last full hour.

TERMINAL OUTPUT MESSAGES
Total number of messages sent by the application to clients, printers or partners during the last full hour.

CURRENT TASKS
: Current number of processes in the application.

CONNECTED USERS
: Number of users connected to the application at the present time.

OPEN DIALOG SERVICES
: Number of dialog services open at the present time.

OPEN ASYN SERVICES
: Number of asynchronous services open at the present time.

CURRENT LOAD
: Current load of the application during the last completed period of 100 seconds as a percentage.

  The value in this field indicates the current load of the processes of the application in processing jobs. If the value is very high, additional processes should be started for the application.

MAXIMUM LOAD
: Maximum load of the UTM application since startup or since the last time the value was reset as a percentage.

DIALOG TAS PER SECOND
: Number of dialog transactions per second being executed at the present time.

ASYN TAS PER SECOND
: Number of asynchronous transactions per second being executed at the present time.

DIALOG STEPS PER SECOND
: Number of dialog steps per second being executed at the present time.

MAXIMUM POOL SIZE
: Maximum allocation of page pool space in percent. In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.

ACTUAL POOL SIZE
: Allocation of page pool space in percent at the present time.

AVERAGE POOL SIZE

Average allocation of page pool space in percent. In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.
For this value to be meaningful, many dialog steps must already have been processed.

CACHE HIT RATE

Hit rate for a page search in cache memory. Figure quoted in percent.
CACHE HIT RATE is reset to 0 before every application start.

NR CACHE SEARCHES

Number of search operations for UTM pages in the cache taken into account to calculate the value of CACHE HIT RATE.

CACHE WAITS FOR BUFFER

Buffer calls in cache that have resulted in a wait time. Figure quoted in percent.
CACHE WAITS FOR BUFFER is reset to 0 after every full hour.

NR CACHE REQUESTS

Number of buffer requests taken into account to calculate the value of CACHE WAITS FOR BUFFER.

UNPROCESSED ATACS

Number of messages for asynchronous services currently stored in openUTM that have not been completely processed.

UNPROCESSED PRINTS

Number of messages currently queued for the printers.

WAITING DPUTS

Number of time-driven jobs currently waiting (DPUTs)

ABNORMAL TERMINATED SERVS

Number of abnormally terminated services. In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.

LOGFILE WRITES

Number of calls written to the user log file (USLOG). The LOGFILE WRITES counter is reset to 0 after every full hour.

UTM-DEADLOCKS

Number of recognized and resolved deadlocks affecting UTM resources. In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.

PERIODIC WRITES

Number of periodic writes since the last application start.
(Periodic write = backup of all log-related administration data on the UTM application).

PAGES PER PERIODIC WRITE

Average number of UTM pages backed up during a periodic write.
The counter is reset to 0 every time the application starts.

WAITS FOR RESOURCES

This value indicates the average lock conflict rate for the GSSB, ULS and TLS memory areas during the last 100 second interval as an amount per thousand, i.e. the total number of wait situations on lock requests as a ratio of GSSB, ULS and TLS lock requests in the last 100 second interval multiplied by 1000.

A high value for WAITS FOR RESOURCES can have the following causes:

- Processes with excessively long run times or wait times,
- Resource disabled for too long, e.g. frequent PEND KP or PGWT calls to KDCS program units.

If a lock holder enters the status PEND KP then all "Waiters" are informed and all further locks are rejected immediately. I.e. the value of WAITS FOR RESOURCES does not increase as a result.

NR RESOURCE REQUESTS

Number of requests for transaction resources during the last 100 second interval taken into account to calculate the value of WAITS FOR RESOURCES.

MAX WAITS FOR RESOURCES

Maximum conflict rate for user data locks across the application run. The value is specified as a percentage

NR RES REQUESTS FOR MAX

Number of requests for transaction resources in the 100 second interval in which the maximum conflict rate MAX WAITS FOR RESOURCES was reached.

WAITS FOR SYSTEM RES

Average conflict rate in the last 100 second interval for the most heavily loaded system resource. The output in different intervals can refer to different system resources. The value is specified as a percentage.

NR SYSTEM RES REQUESTS

Number of requests for system resources during the last 100 second interval taken into account to calculate the value of WAITS FOR SYSTEM RES.

MAX WAITS FOR SYSTEM RES

Maximum conflict rate for requests for system resources (system locks) across the application run. The value is specified as a percentage.

NR SYSTEM RES REQ FOR MAX

Number of requests for system resources in the 100 second interval in which the maximum conflict rate MAX WAITS FOR SYSTEM RES was reached.

ACTUAL JR Only for distributed processing:
Current number of simultaneously addressed job-receiving services relative to generation value MAXJR, figure quoted in percent.
(MAXJR = maximum number of job-receiving services that can be addressed simultaneously in the local application; this corresponds to the number of simultaneously active APRO calls.)

MAXIMUM JR Only for distributed processing:
Maximum number of simultaneously addressed remote job-receiving services relative to generation value MAXJR (KDCDEF control statement UTMD). In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.

The figure is quoted in percent.

AVG COMPRESS PAGES SAVED

Average value for the UTM pages saved per data compression. The writing of data areas in which UTM performs no compression because, for example, the data length is less than one UTM page is not included in this statistics value.

If no statistics value for data compression is available, the string "- - -" is output instead of a numeric value. This is possible in the following situations.

– Data compression is disabled.

– The value was reset, e.g. with KC_MODIFY_OBJECT or by means of WinAdmin or WebAdmin.

– No data compression was performed because the application uses "small" data areas in which it does not make sense to use compression.

If the value output for AVG COMPRESS PAGES SAVED is less than 0.5, for performance reasons data compression should be disabled for this application.

Lifetime of statistical data output for STATISTICS

The following statistical data is updated when the application is started or on each full hour (applies to UTM-F applications as well if MAX STATISTIC-MSG=FULL-HOUR was generated). The following table shows when UTM resets the counter to 0 for a UTM-S application. In UTM-F applications, all counters are reset to 0 at every application start.

You can reset some of the statistical values to 0 through the program interface to administration (see page 365).

Reset time	**Counter(s) reset**
At every application start	CACHE HIT RATE PAGES PER PERIODIC WRITE PERIODIC WRITES
For regeneration with KDCDEF and change generation with KDCDEF/KDCUPD	AVERAGE POOL SIZE MAXIMUM JR MAXIMUM POOL SIZE WAITS FOR RESOURCES
When the application starts and after every full hour (also in UTM-F applications if MAX STATISTIC-MSG= FULL-HOUR was generated)	CACHE WAITS FOR BUFFER LOGFILE WRITES UTM_DEADLOCKS ABNORMAL TERMINATED CONVS TERMINAL INPUT MESSAGES TERMINAL OUTPUT MESSAGES

The following statistical values are written to the system log file SYSLOG at hourly intervals and at every normal termination of the application (message K081) provided that the application has been generated with MAX STATISTICS-MSG=FULL-HOUR:

CACHE HIT RATE
CACHE WAITS FOR BUFFER
CONNECTED USERS
LOGFILE WRITES
TERMINAL INPUT MESSAGES
TERMINAL OUTPUT MESSAGES
UNPROCESSED ATACS

type=SYSLOG

The information in the output is identical to the output from KDCSLOG INFO (see page 779).

type=SYSPARM

```
appliname   APPLINAME                    version   VERSION OF UTM
   ON|OFF   ACCOUNT                       ON|OFF   CALC FOR ACCOUNTING
   ON|OFF   SM2                           ON|OFF   KDCMON
   ON|OFF   TESTMODE                     percent % MAX CACHE PAGING RATE
   number   PROGRAM FGG                  seconds   TERMWAIT
   number   USLOG FGG                    seconds   RESWAIT-TA
   number   MAX TASKS                    seconds   RESWAIT-PR
   number   CURRENT TASKS                seconds   CONRTIME
   number   MAXASYN TASKS                seconds   LOGACKWAIT
   number   CURRENT MAXASYN TASKS         number   CURRENT MAX TASKS IN PGWT
  seconds   PTCTIME                      seconds   CONCTIME
  seconds   PGWTTIME                      number   TASKS WAITING IN PGWT
   YES|NO   PROGRAM EXCHANGE IS RUNNING   number   MAX TASKS IN PGWT
   YES|NO   CLUSTER-APPLICATION           PS|DS    CACHE LOCATION
   ON|OFF   DATA COMPRESSION (GEN)
```

Explanation of the output

APPLINAME Name of the application defined in MAX APPLINAME during KDCDEF generation.

VERSION OF UTM
openUTM version used, including the correction status and generation variant of the application (UTM-S or UTM-F).

ACCOUNT The accounting phase for the accounting function is switched on (ON) or switched off (OFF).
Can be switched on and off during runtime (e.g. with KDCAPPL).

CALC FOR ACCOUNTING
The calculation phase of the accounting function is switched on (ON) or switched off (OFF).
Can be switched on during runtime (e.g. with KDCAPPL).

SM2 Data supply to openSM2 is switched on (ON) or switched off (OFF) for the application.
Can be switched on and off during runtime (e.g. with KDCAPPL).

KDCMON The event monitor KDCMON is switched on (ON) or switched off (OFF).
Can be switched on and off during runtime (e.g. with KDCDIAG).

TESTMODE Test mode is switched on (ON) or switched off (OFF).
This can be switched on and off during runtime (e.g. with KDCDIAG).

MAX CACHE PAGING RATE

Current value for CACHE. The pageing rate indicates the maximum number of pages in cache memory (in percent) to be written to KDCFILE when shortages occur.
This value can be changed, e.g. with KDCAPPL CACHE.

PROGRAM FGG

Generation number of the currently loaded application program.

B
B

For UTM applications on BS2000 systems, the value 0 is always output for PROGRAM FGG.

X/W
X/W
X/W

For UTM applications running on Unix, Linux or Windows systems that were not started from the file generation directory PROG, the value 0 is output for PROGRAM FGG.

TERMWAIT Current value representing the maximum length of time in seconds that can elapse in a multi-step transaction (PEND KP is called in the KDCS program unit) between an output to the terminal and the next entry made by the user (the time the terminal user takes to think).

USLOG FGG Number of the file generation for the user log file to which the user is writing at the present time.

RESWAIT-TA Current value representing the maximum time in seconds that the system can wait for another locked resource (GSSB, ULS,TLS).

MAX TASKS Maximum number of processes allowed in this application (see the *kc_tasks_par_str* data structure as of page 652, *tasks* parameter).

RESWAIT-PR Current value representing the maximum time in seconds that the system can wait for a resource locked by another process (GSSB, ULS, TLS).

CURRENT TASKS

Number of processes in the application at the present time (see the *kc_tasks_par_str* data structure as of page 652, *curr_tasks* parameter).

CONRTIME Current value representing the time in minutes after a connection failure after which UTM should attempt (cyclically) to re-establish the connection.

MAXASYN TASKS

Maximum number of processes in the application that can be used at the same time for asynchronous processing (see the *kc_tasks_par_str* data structure as of page 652, *asyntasks* parameter).

B
B
B
X/W
X/W

LOGACKWAIT

Maximum time in seconds for which openUTM can wait for a printout or transport confirmation message.
For UTM applications running on Unix, Linux or Windows systems, this output is irrelevant.

CURRENT MAXASYN TASKS

Maximum number of processes that can be used at the same time for asynchronous processing (see the *kc_tasks_par_str* data structure as of page 652, *curr_max_asyntasks* parameter).

This value is adjusted automatically whenever:

– the value is explicitly changed by means of system administration functions (e.g. by KDCAPPL ASYNTASKS=).

– the number of processes in the application (CURRENT TASKS) is changed (e.g. by KDCAPPL TASKS=). When the number of CURRENT TASKS is reduced, the number of CURRENT MAXASYN TASKS is also reduced as soon as CURRENT TASKS is < CURRENT MAXASYN TASKS.
If the number of CURRENT TASKS is increased at a later point, the value for CURRENT MAXASYN TASKS is increased again automatically by UTM.

CURRENT MAX TASKS IN PGWT

Maximum number of processes currently in the application that are permitted to accept program units with blocking calls (see the *kc_tasks_par_str* data structure as of page 652, *curr_max_tasks_in_pgwt* parameter).

This value is changed automatically whenever:

– the value is explicitly changed by means of system administration functions (e.g. by KDCAPPL TASKS-IN-PGWT=).

– the number of processes in the application (CURRENT TASKS) is changed by administration (e.g. by KDCAPPL TASKS=). When the number of CURRENT TASKS is reduced, the number of CURRENT MAX TASKS IN PGWT is also reduced as soon as CURRENT TASKS is <= CURRENT MAX TASKS IN PGWT.
If the number of CURRENT TASKS is increased at a later point, the value for CURRENT MAX TASKS IN PGWT is increased again automatically by openUTM.

PTCTIME

Only for distributed processing:
Maximum time in seconds for which a local job-receiving service can wait in PTC mode (prepare to commit, transaction status P) for confirmation from the job-submitting service.
The value 0 signifies that the system can wait for confirmation for an unlimited period of time.

CONCTIME Only for distributed processing:
Time in seconds for monitoring the establishment of a session (LU6.1) or an association (OSI TP). If the session or association is not established within the specified time limit, openUTM terminates the transport connection. This prevents a transport connection from remaining disabled due to failure to establish a session or an association. CONCTIME=0 means that session setup is not monitored in the case of LU6.1 connections (UTM waits indefinitely). In the case of OSI TP connections, UTM waits up to 60 seconds for an association to be set up.

PGWTTIME Maximum time in seconds that a blocking function call can wait, e.g. the KDCS call PGWT.

TASKS WAITING IN PGWT
Number of current processes that can be in wait state at the same time due to blocking function calls (e.g. KDCS call PGWT).

PROGRAM EXCHANGE IS RUNNING
Specifies whether openUTM is currently exchanging a program for the application.

MAX TASKS IN PGWT
Maximum number of processes in the application that can simultaneously process program units with blocking function calls (e.g. KDCS call PGWT) (see the *kc_tasks_par_str* data structure as of page 652, *tasks_in_pgwt* parameter).

CLUSTER-APPLICATION
Specifies whether the application is a UTM cluster application or a standalone UTM application.

CACHE LOCATION
Specifies whether the UTM cache lies in the program space (PS) or in one or more data spaces (DS).

X/W PS is always displayed for Unix, Linux, and Windows systems.

DATA-COMPRESSION (GEN)
Specifies whether data compression is permitted (ON) or not (OFF) for the application. The value displayed here matches the generation value for the application (see openUTM manual “Generating Applications”, MAX DATA-COMPRESSION=). If ON is displayed here, data compression can be enabled or disabled using administration facilities, e.g. with KDCAPPL.

type=TAC

```
TAC  LOCK    STAT TCL    IN-Q   USED   ERROR  DBCNT  TACELAP DBELAP TACCPU D

tac  number  ON   number number number number number msec    msec   mcsec  D
             OFF  type
             HLT
             KP
```

Explanation of the output

TAC	TAC name
LOCK	Lock code which provides access protection for the transaction code; a number between 1 and 4000.
STAT	Status of the transaction code: The TAC is enabled (ON), disabled (OFF), completely disabled (HLT) or blocked (KP). Blocked means that the TAC is disabled, but jobs are accepted for the TAC and placed in the job queue.
TCL	TAC class and type (`D | A | Q`) of the transaction code or TAC queue.
IN-Q	Number of messages that still have to be processed by the program unit started by the TAC name.
USED	Total number of program unit runs processed with this transaction code (only for asynchronous TACs). In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.
ERROR	Number of program unit runs which were started by this transaction code and abnormally terminated. In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.
DBCNT	Average number of database calls from program units which were started with this TAC name. With database connections via the XA interface DBNCT is always 0.
TACELAP	Average run time of the program unit which was started with this TAC (elapsed time); figure quoted in milliseconds.

DBELAP Average time for processing the database calls in the program unit runs. With this TAC; figure quoted in milliseconds.

With database connections via the XA interface DBNCT is always 0.

TACCPU Average CPU time in microseconds used for processing this transaction code in the program unit. This corresponds to the CPU time used by openUTM plus the CPU time used by the database system.

D Indicates whether the transaction code was deleted (D) or not (no entry) from the configuration by means of system administration functions.

The statistical values output for type = USED, ERROR, DBCNT, TACELAP, DBELAP and TACCPU are reset to 0 every time KDCDEF performs a new generation and every time KDCDEF/KDCUPD generates a change in UTM-S applications. With UTM-F applications, the values are set to 0 each time the application is started.

type=TAC-PROG

B *Output for UTM applications on BS2000 systems*

```
TAC       PROGRAM           LOAD-MODULE
tac1      program1          load-module1
tac2      program2          load-module2
tac3      program3          load-module3
```

X/W *Output for UTM applications on Unix, Linux and Windows systems*

```
TAC       PROGRAM           SHARED-OBJECT
tac1      program1          shared-object1
tac2      program2          shared-object2
tac3      program3          shared-object3
```

Explanation of the output

TAC Name of the transaction code

PROGRAM Name of the program unit to which this transaction code is assigned

B LOAD-MODULE
B Name of the load module containing the program unit (PROGRAM)

X/W SHARED-OBJECT
X/W Name of the shared object/DLL containing the program unit (PROGRAM)

type=TACCLASS

```
TACCLASS  TASKS  WT MSGS   AVG-WAIT-TIME PGWT    PRIO* NR     WAITS

    1     number   number  msec          YES|NO  prio  number number
    :
    8     number   number  msec          YES|NO  prio  number number
    9     number   number  msec          YES|NO  prio  number number
    :
   16     number   number  msec          YES|NO  prio  number number
```

*prio = ABS | REL | EQ | NO

Explanation of the output

TASKS — Maximum number of processes currently allowed to process jobs for transaction codes in this TAC class.

WT MSGS — Number of messages currently stored in openUTM for transaction codes in this TAC class that have not yet been processed.

AVG-WAIT-TIME

Average wait time for jobs in this TAC class in milliseconds. The wait time is calculated from the time openUTM accepts the job to the start of actual processing. AVG-WAIT-TIME=0 signifies that all jobs are being processed immediately.

Wait times can, for instance, arise if not all processes in the application may process jobs for the TAC class and if openUTM consequently has to store jobs temporarily in the job queue.

PGWT — Specifies whether program units with blocking calls, e.g. the KDCS call PGWT, are allowed to run in this TAC class.

If the application was generated with priority control (TAC-PRIORITIES statement), then the column PGWT contains NO for all TAC classes.

PRIO — If the application was generated with priority control, then the column PRIO contains the type of priority defined for the TAC classes (ABS, REL, EQ). If the application was generated without the TAC-PRIORITIES statement, the PRIO column contains NO for all TAC classes.

NR — Number of program unit runs for this TAC class.

WAITS — Number of wait situations taken into account to calculate the value AVG-WAIT-TIME.

type=USER

```
USER   KSET   STATUS   OSERV  NR.TACS  CPUTIME   SECCNT   LTERM   D

user1  kset1  ON| OFF  Y| N   number   msec      number   lterm1  D
```

Explanation of the output

USER	Name of the user ID
KSET	Key set assigned to this user ID (access rights)
STATUS	User ID is disabled (OFF) or not disabled (ON).
OSERV	Y signifies that the user is currently processing a service and that this service has reached at least one synchronization point. N signifies that the user is not currently processing a service which has not already reached one synchronization point.
NR.TACS	Number of program unit runs entered under this user ID. In UTM-S applications, the value is set to 0 when the application is generated for the first time with KDCDEF or updated with KDCDEF/KDCUPD. With UTM-F applications, the value is set to 0 each time the application is started.
CPUTIME	Number of CPU milliseconds used up by the user for processing these jobs (does not include the CPU time for the database calls). The value is reset to 0 after the user has signed off (KDCOFF) or after the connection has been cleared.
SECCNT	Number of security violations for this user ID (e.g. incorrect password entered) since the start of the application. This number is reset to 0 each time the application is started.

LTERM The following cases must be distinguished:

– Applications with MULTI-SIGNON=NO (i.e. multiple sign-ons are not permitted):
LTERM or OSI-LPAP partner by means of which a user is signed on with this user ID.
Exception: LTERM contains blanks when signing on to start an asynchronous service was via OSI TP.

– Applications with MULTI-SIGNON=YES (multiple sign-ons permitted):
If a user with the user ID is connected to the application via a terminal, LTERM contains the name of the LTERM partner assigned to the terminal.

If the user ID is generated with RESTART=YES, LTERM contains the name of the LTERM or OSI-LPAP partner via which a client with this user ID is signed on.

Exceptions: Signing on took place by means of OSI TP, and the functional unit “Commit” was selected, or signing on was via OSI TP to start an asynchronous service. In this case, LTERM contains blanks.

In all other cases, LTERM contains blanks.

D Indicates whether the user ID was deleted (D) or not (no entry) from the configuration by means of system administration functions.

The password for this user ID is not output.

KDCLOG - Change the user log file

The user log file USLOG is maintained as the file generation directory USLOG. KDCLOG allows you to close the current user log file (file generation) during live operation and open a new user log file at the same time. This is the file generation with the next generation number in the sequence. The closed log file can then be used in any way you choose. KDCLOG acts on both files if a dual user log file is being used. For further information about the user log file USLOG, please refer to the openUTM manual “Using openUTM Applications”.

Effect in UTM cluster applications

KDCLOG has a global effect in UTM cluster applications, i.e. it applies to all running node applications.

Period of validity of the change

The application writes to the new USLOG file generation(s) until KDCLOG is used to switch to the next file generation.

After the application ends, you can also change to the next file generation using operating system commands (see the openUTM manual “Using openUTM Applications”).

```
KDCLOG
```

For administration using message queuing you must enter KDCLOGA.

KDCLOG has no operands

Output from KDCLOG

The message

```
"COMMAND ACCEPTED"
```

is displayed on the administrator terminal.

KDCLPAP - Administer connections to (OSI-)LPAP partners

KDCLPAP allows you to perform the following actions:

– arrange for connections to be established
– shut down connections
– disable connections or release disabled connections
– define partner applications for which UTM is automatically to establish connections at every application start
– define the number of parallel connections to OSI TP partner applications
– activate replacement connections to OSI TP partner applications - these replacement connections must have been generated with KDCDEF
– change the time for monitoring the idle time modes of sessions and associations

Connections are specified using the name of the LPAP or OSI-LPAP partner to which they are assigned. Replacement connections are identified by the replacement connection name defined in the KDCDEF control statement OSI-CON.

Special issues relating to the establishment and termination of connections

KDCLPAP...,ACT=CON or (CON,number) merely initiates the establishment of a connection. Successful execution of this command does not therefore mean that the connections have in fact been established or that it is possible to establish them (there may be errors in the transport system). You should therefore use KDCINF to check whether a UTM connection can genuinely be established, with the following entry, for example:

```
KDCINF OSI-LPAP, LIST=(osi-lpapname_1,...osi-lpapname_10)   for OSI-LPAP partners
KDCINF CON, LIST=KDCCON                                     for LPAP partners
```

If you wish to establish a connection to a disabled LPAP or OSI-LPAP partner (STATUS=OFF) you must make two KDCLPAP calls:

1. One KDCLPAP call to re-enable the (OSI-)LPAP partner, e.g.:
   `KDCLPAP [OSI-]LPAP=lpapname,STATUS=ON`
2. One KDCLPAP call to ensure that the connection is established, e.g.:
   `KDCLPAP [OSI-]LPAP=lpapname,ACTION=CON`

In conjunction with ACTION=CON **and** STATUS=ON, KDCLPAP will not be processed.

If you wish to reduce the number of parallel connections to an OSI-LPAP partner, call KDCLPAP with ACTION=(CON,number). To do this, you should enter the number of connections you wish to retain in the *number* field.

Effect in UTM cluster applications

The effect in UTM cluster applications is described in the sections devoted to the individual operands since some of the changes made with KDCLPAP apply locally to the node whereas others take effect globally in the cluster.

Period of validity of the change

The period of validity of these changes depends on the type of change and is therefore specified in the description of these operands.

```
KDCLPAP␣   { LPAP    ={ lpapname | (lpapname_1,...,lpapname_10) }           |
             OSI-LPAP={ osi-lpapname | (osi-lpapname_1,..,osi-lpapname_10) }|
             OSI-CON = osi-conname
           }

           [ ,ACTION={ CON| (CON,number) | DIS | ACON | (ACON,number) |
                       NACON | QUIET } ]

           [ ,IDLETIME=time_sec ]

           [ ,STATUS={ ON| OFF } ]
```

For administration using message queuing you must enter KDCLPAPA.

LPAP = (lpapname_1,...,lpapname_10)

Connections to the partner applications to which the LPAP partners *lpapname_1,...,lpapname_10* are assigned are to be administered. For *lpapname_1,...,lpapname_10*, enter the logical names of partner applications generated by the KDCDEF control statement LPAP for distributed processing by LU6.1.

You can enter a maximum of 10 LPAP names for each KDCLPAP call, i.e. you can administer the connections to a maximum of 10 LPAP partners. If you only enter one LPAP name, you do not need to key in the parentheses.

OSI-LPAP = (osi-lpapname_1,...,osi-lpapname_10)

The connections to the partner applications to which the OSI-LPAP partners *osi-lpapname_1,...,osi-lpapname_10* are assigned are to be administered. For *osi-lpapname_1,...,osi-lpapname_10* enter the logical names of partner applications generated by the KDCDEF control statement OSI-LPAP for distributed processing via OSI TP.

You can enter a maximum of 10 OSI-LPAP names for each KDCLPAP call, i.e. you can administer the connections to a maximum of 10 OSI-LPAP partners. If you only enter one OSI-LPAP name, you do not need to key in the parentheses.
If the specified OSI-LPAP is the master LPAP of an OSI-LPAP bundle, it only makes sense to specify STATUS=ON/OFF.

OSI-CON=osi-conname

`KDCLPAP OSI-CON=osi-conname` activates a log connection to an OSI TP partner (OSI-LPAP). The log connection *osi-con* must have been generated statically with the KDCDEF control statement OSI-CON. For *osi-conname is,* enter the name generated in OSI-CON.
You can query the names of all log connections generated for an OSI-LPAP partner with `KDCINF OSI-LPAP`.

When you enter the command, no connections are permitted to the OSI-LPAP partner to which the log connection was assigned for generation. Before UTM activates the log connection, UTM first deactivates the most recent active connection to the OSI-LPAP partner.

The log connection remains active until the end of the application run or until the next time a log connection is selected for the same OSI-LPAP, or until the connection is deactivated.

Entering other operands has no effect. Assignment to the OSI-LPAP partner takes place implicitly using *osi-conname*.

ACTION= ACTION allows you to arrange for connections which were specified in LPAP or OSI-LPAP to be established and shut down. You can define whether or not UTM should automatically establish connections to specified partner applications when the application starts.

CON openUTM arranges for connections to be established to the specified partner applications. All parallel connections generated in the KDCDEF control statement OSI-LPAP for OSI TP partners specified in OSI-LPAP are to be established.

In UTM cluster applications, the operand applies locally in the node.

Successful execution of this command does not mean that the required connections have in fact been established. You can use a KDCINF query to find out whether or not a connection has been established successfully.

If a connection is to be established for a disabled LPAP or OSI-LPAP partner, the partner must be re-enabled with its own KDCLPAP call before the connection is established.

(CON,number)

An entry for *number* is only useful for connections to OSI-LPAP partners. For an LU6.1 partner specified in LPAP, (CON,*number*) acts like CON and the entry for *number* is ignored.

In UTM cluster applications, the operand applies locally in the node.

For OSI-LPAP partners, *number* represents the number of parallel connections to the partner application that are to be established to each of the specified OSI TP partners after the KDCLPAP call. This makes the effect of the call dependent on the entry for *number*. A distinction must be drawn between the following cases:

– If *number* for one of the OSI-LPAP partners specified in OSI-LPAP is greater than the number of parallel connections currently established, openUTM tries to establish the correct number of connections, i.e. the same number of connections to OSI-LPAP partners as specified in *number*.
The maximum number of parallel connections to one OSI-LPAP partner is defined for KDCDEF generation in the OSI-LPAP statement.
If the value for *number* exceeds the generated maximum number of specified OSI-LPAP partners, openUTM only establishes the generated maximum number of connections for this partner.
Successful execution of this command does not mean that the required connections have been established. You can use a KDCINF query to find out whether or not a connection has been established successfully.
If a connection is to be established for a disabled OSI-LPAP partner, the partner must be re-enabled first.

– If *number* is less than the number of parallel connections currently established to a OSI-LPAP partner, UTM shuts down connections to the OSI-LPAP partner until only the number of connections specified in *number* are still in existence.
(CON,0) has the same effect as ACTION=DIS. UTM immediately shuts down all connections for the specified OSI-LPAP partner.

Minimum value of *number*: 0
Maximum value of *number*: Number of parallel connections generated.

DIS Connections to the partners specified in LPAP are shut down immediately. All existing parallel connections to the partners specified in OSI-LPAP are shut down.

In UTM cluster applications, the operand applies locally in the node.

Termination of a connection with DIS has the effect of shutting down all connections immediately. This may cause services to be terminated abnormally. It would be better to use ACTION=QUIET.

ACON (automatic connection)
For the next start and for subsequent starts of this application, UTM is automatically to establish connections to the partner applications specified in LPAP or OSI-LPAP. In the case of OSI TP partners, parallel connections should be established up to the number specified in the appropriate OSI-LPAP statement during KDCDEF generation.

In UTM cluster applications, the operand applies globally in the cluster.

(ACON,number)
Entering *number* is only meaningful for connections to OSI-LPAP partners. In all subsequent starts of the application, UTM should establish connections to the partners specified in OSI-LPAP automatically (automatic connection). The value for *number* defines the number of parallel connections that are to be established to the specified OSI TP partners.

In UTM cluster applications, the operand applies globally in the cluster.

The maximum number of parallel connections to a partner is defined in the OSI-LPAP statement for KDCDEF generation.
If the value in *number* exceeds the generated maximum number for one of the specified OSI-LPAP partners, UTM only establishes the statically generated number of connections for this partner.

For an LU6.1 partner specified in LPAP, (ACON,number) has the same effect as ACON; the entry for *number* is ignored.

Minimum value of number: 0
Maximum value of number: Number of generated parallel connections.

NACON (no automatic connection)
If the ACON property is entered for these connections during generation or by means of system administration functions, it is to be deleted, i.e. openUTM should no longer establish automatic connections to the partner applications specified in LPAP or OSI-LPAP with effect from the next application start.
The entry ACTION=NACON extends beyond the duration of the current application run.

In UTM cluster applications, the operand applies globally in the cluster.

QUIET The connections specified in LPAP or OSI-LPAP are shut down. For OSI-LPAP partners, all parallel connections are shut down.

In UTM cluster applications, the operand applies locally in the node.

With QUIET, connections are not shut down until the sessions or associations generated for the specified LPAP- or OSI-LPAP partners are no longer assigned by dialog or asynchronous jobs. However, no further dialog jobs are accepted for the specified (OSI-)LPAP partners.
The QUIET property can be reset with ACTION=CON.

IDLETIME=time_sec
Time for monitoring the idle state of the sessions or associations generated for the specified LPAP or OSI-LPAP partners.

In UTM cluster applications, the operand applies globally in the cluster.

A change made to IDLETIME remains effective for a defined session or association if it reaches the idle mode next time this command is entered (during establishment of the connection or after completion of the job).
If the session or association is not assigned by a job during the period of time specified in *time_sec*, openUTM shuts down the connection. *time_sec* is defined in seconds.

IDLETIME=0 prevents monitoring of the idle state.

Maximum value: 32767
Minimum value: 60
In the case of values that arre smaller than 60 but not equal to 0, the value 60 is used.

STATUS= Disables or re-enables LPAP or OSI-LPAP partners.

In UTM cluster applications, the operand applies globally in the cluster.

OFF Disables LPAP or OSI-LPAP partners; openUTM does not establish any further connections to this partner application until the LPAP or OSI-LPAP partner is released once again.

No logical connections to the related partner application can be established when an LPAP or OSI-LPAP partner is disabled.

ON Approve LPAP or OSI-LPAP partner again.
This change applies throughout the entire application run.

Output from KDCLPAP

New and old properties are output at the administrator terminal (NEW, OLD).

The following output is produced when `KDCLPAP LPAP=...` is entered:

```
LPAP          STATUS           CONNECTION            IDLETIME

              NEW    OLD       NEW          OLD      NEW    OLD

lpapname      ON     ON        CON  A Q     CON  A Q sec    sec
              OFF    OFF       DIS          DIS
                                  W            W
```

The following output is produced when `KDCLPAP OSI-LPAP=...` is entered:

```
OSI-LPAP      STATUS          CONNECTED          IDLETIME       AUTOCON

              NEW      OLD      NEW      OLD      NEW    OLD    NEW     OLD

osi-lpap      ON  Q    ON  Q    number   number   sec    sec    number  number
              OFF      OFF
```

The following output is produced when `KDCLPAP OSI-CON=...` is entered:

```
OSI-LPAP      OSI-CON

               NEW     OLD

osi-lpap1  osi-con1 osi-con2
```

Explanation of the output

AUTOCON Number of connections to the OSI-LPAP partner that UTM should establish automatically when an application starts.

CONNECTED
Number of parallel connections currently established to the OSI-LPAP partner.

CONNECTION
1st column:
Connection to the LPAP partner is established (CON), shut down (DIS), or UTM is currently trying to establish a connection (W = waiting for connection).

2nd column:
A (automatic) indicates that openUTM will try to establish the connection automatically when the application starts.
Q (quiet) indicates that the connection will be shut down and that no further dialog jobs will be accepted for this partner application.

IDLETIME Monitoring time for the idle state of a session or association on the connection

OSI-CON The name generated with the KDCDEF control statement OSI-CON for the logical connection to the partner application.
osi-con1 is the name of the connection that was active before the changeover; *osi-con2* is the name of the existing replacement connection.

STATUS A connection to the partner application exists or can be established (ON), or cannot be established (OFF).
Q (QUIET) indicates that no further dialog jobs will be accepted for the OSI-LPAP partner and that the connection will be shut down.

KDCLSES - Establish/shut down connections for LU6.1 sessions

With KDCLSES you can arrange for a transport connection to a session to be established or shut down.

Effect in UTM cluster application

In UTM cluster applications, KDCLSES applies locally in the node.

```
KDCLSES␣    LSES=lsesname
           ,ACTION={ CON| DIS | QUIET }
B          [ ,CON=remote_applname,PRONAM=proname,BCAMAPPL=applname ]
X/W        [ ,CON=remote_applname [,PRONAM=proname ],BCAMAPPL=applname ]
```

For administration using message queuing you must enter KDCLSESA.

LSES=lsesname
: Indicates the name of the session requiring administration (local half-session name). For *lsesname*, enter a name which was assigned by means of KC_CREATE_OBJECT for the object type KC_LSES during KDCDEF generation of an LSES statement.

ACTION=
: Controls the establishment and termination of a session.

CON
: A transport connection is to be established for the session *lsesname*. With the operands CON, PRONAM and BCAMAPPL you can specify precisely which transport connection is to be established for the session. If you do not enter a transport connection for CON, PRONAM and BCAMAPPL, UTM tries to establish one of the transport connections generated for the relevant LPAP partner (KC_CREATE_OBJECT for the object type KC_CON or KDCDEF control statement CON).
If openUTM is not able to establish the connection specified in CON, PRONAM and BCAMAPPL, it tries to establish another of the connections generated for the relevant LPAP partner.

DIS
: The connection currently established for the session is shut down immediately.

QUIET
: Connection to the partner application is shut down if the session is no longer assigned by a job.

CON=remote_applname, PRONAM=proname, BCAMAPPL=applname

Entry of this operand is only meaningful for ACTION=CON.

With these operands you can specify the precise transport connection to be established. Your entries must uniquely identify the transport connection. To do so, you must if necessary enter all three operands and make the following entries:

remote_applname
Name of the connection generated for the partner application (remote half-session name assigned dynamically to the partner application by means of KC_CREATE_OBJECT for the object type KC_CON or with the KDCDEF statement CON).

proname
Name of the computer on which the partner application is running.
B This parameter is mandatory on BS2000 systems.

applname
Name of the local application (BCAMAPPL name) via which the connection is established. For *applname*, enter the name that was defined for this application dynamically or in the CON statement during KDCDEF generation.

Output from KDCLSES

The new and old properties (NEW, OLD) of the specified session are output to the administrator terminal.

```
LSES              CONNECTION          CON              PRONAM        BCAMAPPL
                NEW          OLD
lsesname   CON|DIS A|Q  CON|DIS A|Q  remote_applname  proname       applname
```

Explanation of the output

CONNECTION

1st column:
The connection has been established (CON) or shut down (DIS).

2nd column:
A (automatic) indicates that openUTM will try to establish a connection automatically when the application starts.
Q (quiet) indicates that the connection will be shut down and that no further dialog jobs will be accepted for this session.

KDCLTAC - Change the properties of LTACs

The properties of LTACs can be changed with the aid of KDCLTAC. LTACs are the local TAC names for services in partner applications for distributed processing.

Effect in UTM cluster application

In UTM cluster applications, KDCLTAC applies globally to the cluster.

Period of validity of the changes

The changes only apply for the duration of the current application run.

```
KDCLTAC␣   LTAC={ ltacname | (ltacname_1,...,ltacname_10) }
           [ ,STATUS={ ON| OFF } ]
           [ ,WAITTIME=(accesswait_sec[,replywait_sec]) ]
```

For administration using message queuing you must enter KDCLTACA.

LTAC=(ltacname_1,...,ltacname_10)
: Names of the LTACs to be administered. For *ltacname_1,...,ltacname_10,* enter the names of LTACs created dynamically by means of KC_CREATE_OBJECT for the object type KC_LTAC or using the KDCDEF control statement LTAC.

  For each call from KDCLTAC you can enter a maximum of 10 LTAC names. If you only enter one LTAC name you do not need to key in the parentheses.

STATUS =
: Disable LTACs or lift the blocks

  OFF
  : The specified LTACs are disabled and no more jobs are accepted for this LTAC.

  ON
  : The lock is lifted: jobs for the specified LTACs are accepted once again.

WAITTIME=
: Replaces the wait times specified by the generation or administration and replaces them by values specified in *accesswait_sec* and *replywait_sec*.

  accesswait_sec
  : Maximum time in seconds that the system should wait for a session to be reserved (possibly including the establishment of a connection) or for the establishment of an association.

    A wait time of *accesswait_sec* ≠ 0 for asynchronous TACs indicates that the job is always entered in the message queue for the partner application.

A wait time of *accesswait_sec*=0 indicates the following:
In dialog TACs, the service continues in the local application immediately with an appropriate return code if no session or association is available or because the local application is a contention loser (see the KDCDEF control statement SESCHA, LPAP or OSI-LPAP, operand CONTWIN).

With asynchronous TACs, the asynchronous job is rejected with a return code at the FPUT call stage if no logical connection to the partner application exists. If there is a logical connection to the partner application, the message is entered in the output queue.

replywait_sec

Maximum time in seconds that UTM can wait for an answer from the remote service of the partner application.
Restricting the wait time helps to ensure that the wait time for users on the terminal cannot go on indefinitely.

replywait_sec=0 indicates that the system will wait for unrestricted periods of time.

Minimum value: WAITTIME = (0.0) (see above for meaning)
Maximum value: WAITTIME = (32767.32767)

Output from KDCLTAC

The new and old properties of the specified LTACs are output to the administrator terminal.

```
LTAC          STATUS        ACCESSWAIT       REPLYWAIT

            NEW    OLD      NEW     OLD      NEW     OLD

ltacname   ON|OFF ON|OFF    seconds seconds  seconds seconds
```

Explanation of the output

LTAC TAC name of the remote services

STATUS LTAC disabled (OFF) or not (ON)

ACCESSWAIT

Wait time until a session or association is reserved.

REPLYWAIT Wait time for a response to the service program in the partner application

KDCLTERM - Change the properties of LTERM partners

KDCLTERM allows you to change the properties of LTERM partners for clients, printers and LTERM pools. You can disable and enable the LTERM partners and shut down or establish connections to clients and printers.

Effect in UTM cluster applications

The effect in UTM cluster applications is described in the sections devoted to the individual operands since some of the changes made with KDCLTERM apply locally to the node whereas others take effect globally in the cluster.

Period of validity of the change

All changes remain in force after the application has terminated.

```
KDCLTERM␣  LTERM={ ltermname | (ltermname_1,ltermname_2,...,ltermname_10) }
           [ ,ACTION={ CON| DIS } ]
           [ ,STATUS={ ON| OFF } ]
           [ ,PRIMARY=primary-lterm]
```

For administration using message queuing you must enter KDCLTRMA.

LTERM=(ltermname_1,...,ltermname_10)

Name of the LTERM partner to be administered.
For each call from KDCLTERM you can enter a maximum of 10 LTERM names. If you only enter one LTERM partner *ltermname* LTAC name you do not need to key in the parentheses.

For *ltermname_1,...,ltermname_10* you can also enter the names of LTERM partners assigned to an LTERM pool. To do this, enter the full name of this LTERM partner, i.e. the LTERM prefix of the LTERM pool and the serial number.
However, you cannot disable the LTERM partners assigned to LTERM pools, nor can you establish any connections to them. In other words, the only entry you can make for them is ACTION=DIS (connection shutdown).

ACTION= Establishes or shuts down connections to the LTERM partners

In UTM cluster applications, the operand applies locally in the node.

CON This causes connections to be established to the specified LTERM partners. For LTERM partners in an LTERM pool, ACTION=CON is not permitted.

X/W
X/W In UTM applications running on Unix, Linux or Windows systems this function is only available for printers, andTS applications.

DIS Connections to the specified LTERM partners are shut down (DISabled). Connections are shut down immediately, which means that processes cannot be terminated.

STATUS= Disable or enable the LTERM partner.

In UTM cluster applications, the operand applies globally in the cluster.

ON Enable the LTERM partner

OFF Disable the specified LTERM partner.

The block operates as follows:

– A connection request is executed. The connection is established and the following message is issued:
`K027 LTERM partner ltermname disabled – inform administrator or enter KDCOFF`

– An existing connection remains established. Every input with the exception of KDCOFF is acknowledged with the following message:
`K027 LTERM partner ltermname disabled – inform administrator or enter KDCOFF`
The block does not take effect until this connection has reached a synchronization point (end of transaction).

KDCOFF BUT operates for disabled LTERM partners like KDCOFF.

LTERM partner in an LTERM pool cannot be disabled.

PRIMARY=primary-lterm
Name of a normal LTERM or a primary LTERM of an LTERM group (seeopenUTM manual “Generating Applications”).

The PRIMARY operand is only permitted in standalone UTM applications.

The LTERM must be an alias LTERM of an LTERM group. Specifying PRIMARY= causes it to become an alias LTERM of the LTERM group with the primary LTERM *primary-lterm*.

If *primary-lterm* is already the primary LTERM of an LTERM group, the LTERM is assigned to this group. If no alias LTERMs have previously been assigned to *primary-lterm*, and new LTERM group is now created with the *primary-lterm* as the primary LTERM and *ltermname (LTERM=)* as the alias LTERM.

If *primary-lterm* is a normal LTERM, it must meet the following conditions:

– A PTERM with the PTYPE APPLI or SOCKET must be assigned to it.
– It must not be a slave LTERM of an LTERM bundle.
– It must have been generated with USAGE=D.

Output from KDCLTERM

The new and old properties of the LTERM partner (NEW, OLD) are displayed on the administrator terminal. The output ´POOL LTERM´ indicates that the client is connected by means of an LTERM pool.

```
LTERM              STATUS                 CONNECTION

               NEW         OLD          NEW          OLD

ltermname   ON | OFF    ON | OFF   CON| DIS| W  CON| DIS| W [POOL LTERM]
     :          :           :            :            :                :
```

Explanation of the output

LTERM Name of the LTERM partner

STATUS The LTERM partner is disabled (OFF) or not disabled (ON).

CONNECTION
Connection to the LTERM partner is established (CON) or shut down (DIS), or openUTM is currently trying to establish a connection (W = waiting for a connection).

KDCMUX - Change properties of multiplex connections (BS2000 systems)

KDCMUX allows you to change the properties of multiplex connections. You can:

– disable or re-enable multiplex connections
– activate or deactivate multiplex connections
– make specifications regarding the establishment of a connection when the application starts.

Things to note when establishing a connection

KDCMUX...,ACTION=CON merely initiates the establishment of a connection. Successful execution of this command does not therefore mean that the connections are actually established or even that they can, in fact, be established successfully (e.g. a connection attempt may fail due to a fault in the transport system). You should therefore use KDCINF to check whether openUTM has actually been able to establish a connection. For example:

```
KDCINF MUX,LIST=(muxname_1,muxname_2,...,muxname_10)
```

If a connection is to be established for a disabled multiplex connection (STATUS=OFF), you must make two calls with KDCMUX.

1. KDCMUX to re-enable the multiplex connection, e.g.:
   ```
   KDCMUX MUX=muxname,STATUS=ON
   ```
2. KDCMUX call to arrange for a connection to be established, e.g.:
   ```
   KDCMUX MUX=muxname,ACTION=CON
   ```

Effect in UTM cluster applications

The effect in UTM cluster applications is described in the sections devoted to the individual operands since some of the changes made with KDCMUX apply locally to the node whereas others take effect globally in the cluster.

Period of validity of the change

The period for which these changes remain valid depends on the type of change and is therefore specified in the description of each operand.

```
KDCMUX␣    MUX={ muxname | (muxname_1,muxname_2,...,muxname_10) }
           [ ,ACTION={ CON| DIS | ACON | NACON } ]
           [ ,BCAMAPPL=applname ]
           [ ,MAXSES=number_sessions ]
           ,PRONAM=proname
           [ ,STATUS={ ON| OFF } ]
```

For administration using message queuing you must enter KDCMUXA.

MUX=(muxname_1,muxname_2,...,muxname_10)

Name of the multiplex connection to be administered.
For *muxname_1,...,muxname_10*, names must be entered that have been defined with MUX statements during KDCDEF generation.
For each KDCMUX call you can enter a maximum of 10 names. If you only enter one name you do not need to key in the parentheses.

ACTION=

ACTION allows you to initiate the establishment and shutdown of connections to the specified multiplex connections. You can specify whether or not openUTM is to establish connections to the specified multiplex connections automatically in subsequent application starts.

CON

(connection)
openUTM initiates the establishment of connections to the specified multiplex connections.

In UTM cluster applications, the operand applies locally in the node.

KDCINF allows you to check whether or not the connection was established successfully.

DIS

(disconnection)
Connections to the specified multiplex connections are shut down. The connection is shut down with immediate effect: this means that not even open services can be completed.

In UTM cluster applications, the operand applies locally in the node.

ACON

(automatic connection)
At subsequent application starts, openUTM is to activate the multiplex connections automatically, i.e. it is to establish the connections automatically.

In UTM cluster applications, the operand applies globally in the cluster.

B ACTION=ACON takes effect until automatic connection establishment is explicitly reset by means of system administration functions (NACON action).

B NACON (no automatic connection)
B If the ACON property is entered for the specified multiplex connections during generation by the administration function, it is to be deleted. In other words connections to the specified multiplex connections should no longer be established automatically during subsequent application starts.

B In UTM cluster applications, the operand applies globally in the cluster.

B BCAMAPPL=applname
B Name of the local application through which connections are established to the multiplex connections. For *applname*, the application name assigned to the multiplex connections in the MUX statements during KDCDEF generation should be specified, i.e. the name which the message router must pass to UTM in order for a connection to be established.
B Default value:
B If you do not enter BCAMAPPL, the name generated in the APPLINAME operand in the KDCDEF control statement MAX is assumed.

B MAXSES=number_sessions
B *number_sessions* defines the maximum number of terminals through which each of these multiplex connections can be connected at the same time.

B In UTM cluster applications, the operand applies globally in the cluster.

B This change only applies to the current application run.

B Minimum value: 1
B Maximum value: 65000

B PRONAM=proname
B Name of the processor running the message router to which the multiplex connection is assigned.

STATUS= The specified multiplex connections are disabled or released again.

In UTM cluster applications, the operand applies globally in the cluster.

ON Releases (enables) the multiplex connections (with immediate effect).

OFF Disables multiplex connections. No connection to any of the specified multiplex connections should exist at the point in time when they are disabled.
If such a connection does exist, openUTM will not disable it. The value ON is issued in the output from KDCMUX (see page 761) for STATUS NEW and STATUS OLD.

Output from KDCMUX

The new and old properties of the multiplex connections (NEW, OLD) currently being administered are displayed on the administrator terminal.

MUX	PRONAM	BCAMAPPL	STATUS		CONNECTION				MAXSES	
			NEW	OLD	NEW		OLD		NEW	OLD
muxname :	proname :	applname :	ON| OFF :	ON| OFF :	C| D| W :	A	C| D| W :	A	number :	number :

Explanation of the output

MUX Name of the multiplex connection

PRONAM Name of the processor on which the message router is running

BCAMAPPL Name of the UTM application which was assigned to the multiplex connection for KDCDEF generation.

STATUS The multiplex connection is disabled (OFF) or not disabled (ON).

CONNECTION

1st column:
The multiplex connection is connected to the application (C) or not (D), or openUTM is currently trying to establish a connection to the multiplex connection (W = waiting for a connection).

2nd column:
Every time the application starts, openUTM will try to establish a connection automatically (A) or not (no entry).

MAXSES Maximum number of terminals that can be connected to the application via this multiplex connection at the same time.

KDCPOOL - Administer LTERM pools

KDCPOOL allows you to redefine the number of enabled and/or disabled clients for an LTERM pool.

Effect in UTM cluster applications

In UTM cluster applications, KDCPOOL applies globally to the cluster.

Period of validity of the change

The change remains in force after the application has terminated.

```
KDCPOOL␣      LTERM=ltermprefix
           [ ,STATUS=( { ON| OFF }, number_clients ) ]
```

For administration using message queuing you must enter KDCPOOLA.

LTERM=ltermprefix
: LTERM prefix of the LTERM pool, as generated in the KDCDEF control statement TPOOL. If you enter *ltermprefix*, the LTERM pool to be administered is identified uniquely.

STATUS=
: Defines the number of clients able to connect via the LTERM pool at the same time. The maximum number of clients able to connect via the LTERM pool at the same time is defined during KDCDEF generation (NUMBER in the control statement TPOOL). Using the administration function you can reduce this number or increase a number that has previously been reduced back to the maximum number.

(ON,number_clients)
: *number_clients* defines the number of approved LTERM partners for the LTERM pool.

(OFF,number_clients)
: *number_clients* defines the number of disabled LTERM partners in the LTERM pool, i.e. the maximum number of LTERM partners defined during KDCDEF generation is reduced by *number_clients*.

  Locks assigned to LTERM partners in an LTERM pool operate as follows:

  – UTM rejects a connection request from a client if the permissible number of LTERM partners for that LTERM pool has already been reserved by other clients.

– If, at the time the command is processed, more connections to the LTERM pool exist than the permissible number of LTERM partners, then all existing connections are initially retained.
The lock only takes effect after the connection has been shut down if a client has placed a new communication request.
If terminal users sign off with KDCOFF BUI they can sign back on with KDCSIGN even if, at this time, more than the permissible number of LTERM partners in the LTERM pool are still reserved.

Minimum value of *number_clients*: 0

Maximum value of *number_clients*:
The maximum number of clients specified for KDCDEF generation which can connect at the same time through this LTERM pool. If, when clients are approved (ON,*number_clients*) *number_clients* is greater than the maximum value, UTM automatically reduces the value for *number_clients* to the maximum value.

Output from KDCPOOL

The new and old number of clients enabled for the LTERM pool is output to the administrator terminal in the following form:

```
POOL          PRONAM     BCAMAPPL   PTYPE            STA=ON
                                                  NEW        OLD
ltermprefix   proname     applname   ptype      number     number
```

Explanation of output

POOL	LTERM prefix generated for the LTERM pool
PRONAM	Name of the computer to which the LTERM pool was assigned
BCAMAPPL	Name of the UTM application assigned to the LTERM pool during KDCDEF generation
PTYPE	Type of clients entitled to connect via the LTERM pool
STA=ON	Number of LTERM partners approved for the LTERM pool before the command was processed (OLD) and the number of LTERM partners currently approved for the LTERM pool (NEW).

KDCPROG - Replace load modules/shared objects/DLLs

KDCPROG allows you to use the BLS interface to replace load modules in a UTM application on a BS2000 system, or to replace shared objects in a UTM application on Unix, Linux and Windows systems if you are working with the function “Program exchange with shared objects”. See also the openUTM manual “Generating Applications” and the corresponding openUTM manual “Using openUTM Applications”.

W W W On Windows systems, shared objects are realized using DLLs. Details pertaining to handling the DLLs are also described in the openUTM manual “Using openUTM Applications on Unix, Linux and Windows Systems”.

Requirements for program replacement using KDCPROG

You can replace or reload sections of an application program if they satisfy the following parameters:

- The program sections to be replaced must have been generated as separate load modules/shared objects/DLLs.
- Every load module or shared object/DLL to be replaced must have been generated statically using a LOAD-MODULE statement (BS2000 systems) or a SHARED OBJECTS statement (Unix, Linux and Windows systems).
- The load modules/shared objects/DLLs must not have been statically linked to the application program.

B B
- The load modules to be replaced must not have been loaded in system memory (class 4 memory), nor in a global common memory pool (generated with SCOPE=GLOBAL).

To enable openUTM to process the command, a load module/shared object/DLL must exist with the specified name and version defined in *version* in the program library or directory that was assigned to it during KDCDEF generation:

B
- LOAD-MODULE statement, operand LIB (BS2000) or

X/W X/W
- SHARED-OBJECT statement, operand DIRECTORY (Unix, Linux and Windows systems).

If, in violation of this rule, no load module/shared object/DLL with the version defined in *version* exists in this program library or in the specified directory, the previously loaded load module/shared object/DLL will be unloaded and a message issued.

If you then call KDCPROG again, you can load the load module/shared/DLL object by specifying a version of the load module/shared object that already exists in the library in *version*, or by placing the missing load module/shared object/DLL with the specified version in the program library/program directory.

How to implement a program replacement

The way a program replacement is implemented depends on the generated load mode of the load module/shared object/DLL.

You generate the load mode in the LOAD-MODULE statement (BS2000 systems) or in the SHARED-OBJECT statement (Unix, Linux and Windows systems), in both cases in the LOAD-MODE operand.

- LOAD-MODE=STARTUP
  (The load module/shared object/DLL is loaded as a separate unit when the application starts.)
  This replacement operation is performed for each process before the next job is processed, without first terminating the current application program. Several processes in the application can be terminated simultaneously. Until the program replacement has been completed for all processes in the application, you are not allowed to initiate any further replacements with KDCPROG.

  The KDCPROG call merely initiates a program replacement. The actual process of program replacement can take some considerable time. openUTM informs you of the success or failure of the program replacement operation with messages to SYSOUT and SYSLOG (BS2000 systems) or *stdout* and *stderr* (Unix, Linux and Windows systems).

- LOAD-MODE=ONCALL
  (Loaded whenever a program unit is called up out of the load module/shared object/DLL for the first time)
  Replacement is performed for every process even if a program unit from this load module/shared object is called next time in the same process. At any given time, several processes in the application can be replaced simultaneously.

B
B
B
B
B
B
B
B
B
B
B
B
B
- LOAD-MODE=(POOL, POOL/STARTUP, POOL/ONCALL)
  (The public slice of the load module is loaded into a common memory pool.)
  In standalone UTM applications, program replacement operation is not initiated by the KDCPROG call. KDCPROG only causes the new version to be flagged. The new version of the load module is not loaded into the common memory pool until the following replacement of the entire application program with KDCAPPL PROG=NEW. In this event, another KDCPROG call can follow immediately after the call from KDCPROG.
  You can flag several load modules using several KDCPROG calls which are then replaced in response to the next KDCAPPL PROG=NEW. If no KDCAPPL PROG=NEW follows in the same application run, the flagged versions are replaced after the next application start.

B
B
  In UTM cluster applications, openUTM automatically initiates program replacement when you modify the version of a load module in the Common Memory Pool.

Effect in UTM cluster applications

In UTM cluster applications, KDCPROG applies globally to the cluster. Program replacement is performed in all running node applications.

Period of validity for a program replacement

The change remains in force after the end of the application.

```
KDCPROG␣   VERSION=version
B          , LOAD-MODULE=lmodname
X/W        , SHARED-OBJECT=shared-object-name
```

For administration using message queuing you must enter KDCPROGA.

VERSION=version

Version of the load module/shared object/DLL which has to be loaded. The value for *version* must not exceed 24 characters in length.

B In UTM applications on BS2000 systems you must always specify the next version of the load module to be loaded.
When the replacement is initiated, the library to which the load module was assigned for KDCDEF generation (see also KDCDEF statement LOAD-MODULE...,LIB=) must contain an element with the name *lmodname* and the version specified in *version*.
You cannot replace load modules that have been linked statically to the application program (load mode STATIC).
Similarly, load modules which have the STARTUP load mode and contain TCB entries can also not be replaced.

X/W In UTM applications running on Unix, Linux or Windows systems you must enter the version name if the shared object was generated with the ONCALL load mode.
Entering the version name is optional for shared objects/DLLs with the STARTUP load mode if you are not using the version concept.

B LOAD-MODULE=lmodname

B Name of the load module to be replaced. This can be the name of an OM or an LLM. The load module (with this name) must have been configured for KDCDEF generation with a LOAD-MODULE statement. You can only enter one name for each KDCPROG call.
The name must not be more than 32 characters long.

X/W SHARED-OBJECT=shared-object-name

Name of the shared object/DLL to be replaced. The name must have been generated with a SHARED-OBJECT statement. For each KDCPROG call you can only specify one name.
The name must not be more than 32 characters long.

Output from KDCPROG

After a KDCPROG call is placed, the following information is output to the administrator terminal:

B *Output for UTM applications on BS2000 systems*

```
LOAD-MODULE          lmodname
VERSION (PREVIOUS)   old element version
VERSION (CURRENT)    new element version
LIBRARY              name of program library
LOAD MODE            STARTUP |? ONCALL |? POOL | POOL/STARTUP | POOL/ONCALL
```

X/W *Output for UTM applications running on Unix, Linux or Windows systems*

```
SHARED-OBJECT        shared object name
VERSION (PREVIOUS)   old  version
VERSION (CURRENT)    new  version
DIRECTORY            name of program directory
LOAD MODE            STARTUP |? ONCALL ?
```

Explanation of the output

B LOAD-MODULE

Name of the load module

X/W SHARED-OBJECT

Name of the shared objects/DLL

VERSION (PREVIOUS)

Previously loaded version of the load module/shared object/DLL

VERSION (CURRENT)

Version of the load module/shared object which is to be loaded

B LIBRARY Name of the program library from which the load module (BS2000 systems) is loaded.

X/W DIRECTORY Name of the directory from which the shared object/DLL is loaded.

LOAD MODE Load mode for the load module/shared object/DLL:

STARTUP
The load module/shared object/DLL is loaded as a separate unit when the application starts.

ONCALL
The load module/shared object/DLL is loaded whenever a program unit is called from the load module for the first time.

B
B
B
POOL
The load module is loaded into the common memory pool when the application starts. The load module does not contain a private slice.

B
B
B
B
POOL/STARTUP
The public slice of the load module is loaded into the common memory pool when the application starts. The private slice belonging to the load module is then loaded into the local process memory.

B
B
B
B
B
POOL/ONCALL
The public slice of the load module is loaded into the common memory pool when the application starts. The private slice belonging to the load module is loaded into the local process memory when the first program unit assigned to this load module is called.

Program replacement messages with KDCPROG

Once the replacement of application units generated by a STARTUP (and POOL with BS2000 systems) load mode is complete, the following message is output to SYSOUT and SYSLOG (BS2000 system) or *stdout* and *stderr* (Unix, Linux or Windows system):

```
K074 Replacement of the application programs was performed
```

If errors occur when application units generated with STARTUP (and POOL with BS2000 systems) load mode are being replaced, the following message is output to SYSLST and SYSLOG (BS2000 system) or *stdout* and *stderr* (Unix, Linux or Windows system):

```
K075 Replacement of the application program was not performed due to errors
```

B
B
If errors occur during replacement, message K078 together with the error cause is output to SYSOUT.

KDCPTERM - Change properties of clients and printers

KDCPTERM allows you to change the properties of clients and printers.

You can perform the following actions:

- Disable or re-enable clients and printers.
- Establish and shut down logical connections to clients and printers. In particular, you can establish or shut down connections to individual printers in a printer pool.
- Initiate or prevent the automatic establishment of connections to clients and printers when the application starts.

B B
- If terminals are connected to the application via a multiplex connection, you can release sessions that are in DISCONNECT-PENDING mode.

Things to note when establishing and shutting down connections

With KDCPTERM you can initiate immediate connections or the automatic establishment of a connection for each subsequent application start for the following objects (ACTION=CON or ACON):

B B B B
- Printers, terminals and transport system applications of the type APPLI or SOCKET.
  Calls from connections to UTM clients with the UPIC carrier system (PTYPE=UPIC-R) are rejected.
  The initiative for establishing a connection always lies with the UTM client.

X/W X/W X/W X/W X/W
- Printers (PTYPE=PRINTER) and transport system applications of the type PTYPE=APPLI or SOCKET.
  Calls from connections to UPIC clients (PTYPE=UPIC-R/-L) and to terminals (PTYPE=TTY) are rejected. The establishment of connections to these clients can only be initiated by the clients themselves.

No connections can be established with KDCPTERM to clients which connect to the application by means of an LTERM pool.

If there is a request for connection to be established with a client for which the actions CON and ACON are not permitted, the KDCPTERM call is rejected.

In response to a successful call from KDCPTERM with the action CON, UTM initiates the establishment of a connection to the specified clients and printers. Successful execution of this command does not mean that the connections have actually been established or even, indeed, that they can be established. To find out whether the connections are actually possible, you must enter a specific query (e.g. with KDCINF).

Termination of a connection with ACTION=DIS causes the connection to a client or printer to be shut down immediately. Neither can open services be terminated.

Things to note when disabling clients or printers

A lock operates as follows:

– Every connection request from a client is rejected.
– Existing connections are retained.
  The lock only comes into effect when a client next attempts to establish a logical connection.
– Calls from a connection to a disabled client or printer are rejected.

Asynchronous messages to disabled clients or printers are stored in the buffer memory or the KDCFILE and can give rise to a shortage or resources!

Effect in UTM cluster applications

The effect in UTM cluster applications is described in the sections devoted to the individual operands since some of the changes made with KDCPTERM apply locally to the node whereas others take effect globally in the cluster.

Period of validity of changes

The period during which the changes remain effective is dependent on the type of change and is therefore specified in the operand descriptions.

```
KDCPTERM␣  PTERM={ ptermname | (ptermname_1,ptermname_2,...,ptermname_10) }
           [ ,BCAMAPPL=applname ]
           [ ,STATUS={ ON| OFF } ]
B            ,PRONAM=proname
B          [ ,ACTION={ CON| DIS | REL | ACON | NACON } ]
X/W        [ ,PRONAM=proname ]
X/W        [ ,ACTION={ CON| DIS | ACON | NACON } ]
```

For administration using message queuing you must enter the KDCPTRMA administration command.

PTERM=(ptermname_1,...,ptermname_10)

Name of the clients and printers to be administered. You can enter a maximum of 10 names for each KDCPTERM call.
If only one name is entered you do not need to key in the parentheses.

All names on the list must belong to clients and printers located on the same computer.

BCAMAPPL=applname

Only applicable to client applications of the PTYPE=APPLI/SOCKET or PTYPE=UPIC-R/L type.

For *applname*, enter the name of the local UTM application with which the connections between the UTM application and client applications are established.

Default: The application name specified in MAX APPLINAME for KDCDEF generation is accepted.

STATUS= Disables a client or printer or allows disabled clients and printers to be used again (i.e. to be re-enabled).

In UTM cluster applications, the operand applies globally in the cluster.

ON The clients/printers *ptermname_1,...,ptermname_10* are released for use again.

OFF The clients/printers *ptermname_1,...,ptermname_10* are to be disabled.

Enabling and disabling in this manner extend beyond the end of the current application.

PRONAM=proname

Name of the processor on which the clients/printers *(ptermname_1,...,ptermname_10)* are located. At this point, enter the name of the processor that was specified when the clients/printers were specified in the configuration.

B
B

For clients and printer in a UTM application on BS2000 systems, entering *proname* is mandatory.

X/W
X/W

In a UTM application running on a Unix, Linux or Windows system you do not enter *proname* if the specified clients and printers are connected locally.

ACTION= Defines which action openUTM is to perform.

CON openUTM should establish logical connections to the clients and printers *ptermname_1,...,ptermname_10*.

In UTM cluster applications, the operand applies locally in the node.

ACTION=CON is **not** permitted for:

– UPIC clients
– clients connected to the application by an LTERM pool

B
B
X/W
X/W

– terminals connected to the openUTM application on BS2000 systems by a multiplex connection
– terminals in a UTM application on a Unix, Linux or Windows system (PTYPE=TTY)

DIS

openUTM is to shut down logical connections to the clients and printers *ptermname_1,...,ptermname_10*.

In UTM cluster applications, the operand applies locally in the node.

B
B
B
B
B
B
B

In openUTM on BS2000 systems, ACTION=DIS is rejected if a terminal connection is to be shut down which is connected to the UTM application by a multiplex connection and which exists for a session in DISCONNECT-PENDING mode. The KDCPTERM call is rejected. The logical connection must be shut down with ACTION=REL.
You can check whether or not a session is in DISCONNECT-PENDING mode with KDCINF PTERM.

B
B
B

REL

ACTION=REL is only permitted if the clients specified in *ptermname_1,...,ptermname_10* are connected to the application by a multiplex connection.

B

In UTM cluster applications, the operand applies locally in the node.

B
B
B
B

A session is released with ACTION=REL. The logical connection to the client is shut down.
Entry of ACTION=REL is only permitted if the session is in DISCONNECT-PENDING mode and if its timer has run out (approx. 10 minutes).

ACON

(automatic connection)
During subsequent application starts, UTM should automatically establish logical connections to *ptermname_1,...,ptermname_10*.

In UTM cluster applications, the operand applies globally in the cluster.

ACTION=ACON is **not** permitted for:

– UPIC clients
– clients connected to the application via an LTERM pool

X/W
X/W

– terminals on a UTM application on a Unix, Linux or Windows system with PTYPE=TTY.

NACON

(no automatic connection)
Renders the ACON entry ineffective, i.e. openUTM does not establish any logical connections to *ptermname_1,...,ptermname_10* during subsequent application starts.

In UTM cluster applications, the operand applies globally in the cluster.

Output from KDCPTERM

The new and old properties of specified physical clients and printers (NEW, OLD) are displayed on the administrator terminal.

```
PTERM   PRONAM   BCAMAPPL      STATUS            CONNECTION
                              NEW      OLD       NEW         OLD
pterm1  proname  applname ON|OFF  ON|OFF  C|D|W  A M  C|D|W  A M  [POOL PTERM]
                                         T|E        T|E
```

Explanation of the output

PTERM Name of the client/printer

PRONAM Name of the processor on which the client/printer is located

BCAMAPPL Name of the local UTM application through which connections are established to the client/printer

STATUS The client/printer is disabled (OFF) or not disabled (ON)

CONNECTION

1st column:

C/D/W Client/printer is currently connected to the application (C) or is not connected (D), or openUTM is trying to establish a connection (W = waiting for a connection)

B T/E Is only output for terminals which use a multiplex connection to connect with a UTM application on a BS2000 system.
T: (timer)
The session is in DISCONNECT-PENDING mode; the timer is running, waiting on confirmation that a connection has been established
E: (expired)
The session is in DISCONNECT-PENDING mode and has timed out waiting for confirmation.

2nd column:
A: Connection established automatically when the application starts

B 3rd column (only for UTM applications on BS2000 systems):
The client is connected (M) or not (no entry) to the application via a multiplex connection.

POOL PTERM

This is output if the client is connected via LTERM pool.

KDCSEND - Send a message to LTERM partners (BS2000 systems)

KDCSEND allows you to send messages to one, several or all active terminals of a UTM application on BS2000 systems. openUTM then sends message K023 with the specified messages as an insert. This is output by default in the system line on the terminal. However, the message destination of message K023 can be changed. If the message destination PARTNER is selected for the UTM message K023, you can send the message to one, several or all connected TS applications. The message goes only go to dialog partners (LTERM with USAGE=D).

Effect in UTM cluster applications

In UTM cluster applications, KDCSEND applies locally in the node.

```
KDCSEND␣    MSG=´message´
          [ ,LTERM={ ltermname | (ltermname_1,...,ltermname_10) | KDCALL } ]
```

For administration using message queuing you must enter KDCSENDA.

MSG=´message´
: For *message*, enter the message to be sent. It should be enclosed in single quotes and must not be longer than 74 characters. Write double quotes within the message text (i.e. do not use single inverted commas as part of your message text).
If a terminal is assigned to the LTERM partner, the message is displayed in the system line.

LTERM=
: Specifies the LTERM partner to which the message should be sent.

(ltermname_1,...,ltermname_10)
: Name of the LTERM partner to which the message is to be sent. You can enter a total of up to 10 names. If you only enter one name, you do not need to key in the parentheses.

KDCALL
: This message should be sent to all active LTERM partners, i.e. to all clients connected by a logical connection at the present time.

Default: KDCALL

Output from KDCSEND

The message *message* is displayed on the administrator terminal.

KDCSHUT - Terminate an application run

KDCSHUT allows you to terminate a UTM application. You have the following options:

In UTM cluster applications, you can specify whether the application run is to be terminated at all nodes or only at the node at which the call is issued.

You have the following options:

- You can terminate the application run normally. UTM terminates the application run as soon as all running dialog steps have been completed (NORMAL).
- You can schedule the application to terminate after a specified period (WARN).
- You can terminate the application once all the UTM-D dialogs have been terminated and all the UTM-D connections have been disconnected and at the latest, however, after a specified period (GRACE).
- You can kill the application, i.e. perform an immediate abnormal termination (KILL).

You should note the following if you kill an application:

You cannot kill the application by means of an asynchronous service, i.e. the asynchronous transaction code KDCSHUTA KILL has no effect.

You should note the following when shutting down applications with distributed processing:

- You should preferably terminate applications with distributed processing with KDCSHUT GRACE or alternatively with a warning (KDCSHUT WARN).
  The use of KDCSHUT GRACE or WARN reduces the probability that services will be killed and distributed transactions will remain in transaction status P (preliminary end of transaction).
- An application with distributed processing is not terminated normally if, at shutdown time, there are still services with transaction status P (prepare to commit) or if acknowledgments are still outstanding for asynchronous messages to a partner server. In such cases, openUTM issues message K060 with ENDPET as the reason for termination. No dumps are generated.

  Consequently, for KDCSHUT WARN or GRACE, you should specify a time that is greater than the maximum period that a distributed transaction remains in the state PTC (i.e. transaction status P). This reduces the probability of distributed transactions still being in this state at the end of the application and of the application being terminated abnormally with ENDPET.

For further information about shutting down/terminating a UTM application, please refer to the openUTM manual “Using openUTM Applications”..

```
KDCSHUT␣      { GRACE [ ,TIME=time_min ] | KILL | NORMAL |
                WARN [ ,TIME=time_min ] }

              [ ,SCOPE= { LOCAL | GLOBAL } ]
```

For administration using message queuing you must enter KDCSHUTA.

GRACE All the active LPAP and OSI-LPAP connections are set to QUIET. The application is terminated as soon as all the UTM-D connections have been disconnected or, at the latest, when the defined time has expired.

B
B
B

At all active terminals, a note in the system line indicates the impending shutdown of the application. This is accompanied by an indication of the time remaining before shutdown (see TIME operand).

After KDCSHUT GRACE has been entered, only users with administration authorizations may sign-on. It is then only possible to start services whose service TAC belongs to an administration program unit. All UTM user commands with the exception of KDCOUT are still executed.

The application run is terminated, i.e. it is shut down immediately. Open services will not be terminated first. A UTM dump with the dump code=´ASIS99´ is created by all processes.

NORMAL Termination of the application is initiated immediately. No more users can sign on to the application and users cannot start any new services. No new dialog entries are processed. If the new dialog entry is a multi-step transaction, the multi-step transaction is rolled back to the last synchronization point. All logical connections to clients and printers are shut down.
Users can continue working on open services after the next application start.

WARN All active connections from LPAPs and OSI-LPAPs are set to QUIET.

B
B
B

At all active terminals, a note in the system line indicates the impending shutdown of the application. This is accompanied by an indication of the time remaining before shutdown (see TIME operand).

After KDCSHUT WARN has been entered, only users with administration authorizations may sign-on. It is then only possible to start services whose service TAC belongs to an administration program unit. All UTM user commands with the exception of KDCOUT are still executed.

TIME=time_min

Only works together with WARN and GRACEFUL.

Meaning for GRACE:
time_min is the maximum time in minutes after which the application will be terminated.

Meaning with WARN:
time_min is the time in minutes after which the application is terminated.

Default: 10 minutes
Maximum: 255 minutes
Minimum: 1 minute

openUTM rejects the entry TIME=0.

B At all active terminals, a note in the system line indicates the impending shutdown of the application. This is accompanied by an indication of the time remaining before shutdown. If a very large number of terminals are active (configurations with many terminals) then it takes a certain amount of time to issue the shutdown notification. You should therefore not choose too short a value for TIME.

B During KDCDEF generation, TAC KDCSHUT was assigned the maximum CPU time required by KDCSHUT to perform a shutdown. Select a sufficiently long period of time for applications involving numerous terminals. If this period of time is not long enough, openUTM terminates the process and issues the message K017.

SCOPE= Specifies the scope of application of the command.
This parameter is only relevant for UTM cluster applications.

LOCAL The command only applies to local node applications.
Default value.

GLOBAL The command applies to all the node applications in the UTM cluster application. SCOPE=GLOBAL is rejected if the running node applications have not all been generated in the same way. This may occur, for example, if an update generation of the KDCFILE is performed without fully shutting down the UTM cluster application.

Output from KDCSHUT

The message "COMMAND ACCEPTED" is displayed on the administrator terminal. UTM displays the actual termination of the application in the following manner:

B B
- The end of the application is only displayed on the console. The display appears as soon as the last process in the UTM application has finished.

X
- The end of the application is logged by the *utmmain* process after *stdout* and *stderr*.

W W W
- The end of the application is logged in *stdout* and *stderr* by the process *utmmain*. If the application is started as a service, messages are also entered in the event logfile of the Windows system.

KDCSLOG - Administer the SYSLOG file

With KDCSLOG you can administer the system log file SYSLOG during runtime. You can perform the following activities:

– Switch automatic size monitoring of the SYSLOG on and off.
– Define or change the control value for size monitoring.
– Switch the SYSLOG file to the next file generation of the SYSLOG-FGG.
– Write the contents of the internal UTM message buffer to the SYSLOG file.
– Call for information about the properties of the SYSLOG file.

Effect in UTM cluster applications

The call applies globally to the cluster, i.e. the system log file SYSLOG is administered for each node application. Size monitoring extends beyond the current UTM cluster application run. Switching or writing of the buffer apply only to the current UTM cluster application run, i.e. to all the node applications that are currently running.

Period of validity of the change

The most recent control value set for size monitoring is also set after the next application start.

If the SYSLOG-FGG basis falls within the valid range of the SYSLOG-FGG (between the first and last file generations), openUTM first logs the basic file generation in the next application run. If the basis falls outside the valid range, openUTM opens a new file generation for the log.

```
KDCSLOG␣   { INFO | WRITE | SIZE=fg_size | SWITCH [ , SIZE=fg_size ] }
```

For administration using message queuing you must enter KDCSLOGA.

INFO — Information about the SYSLOG file or SYSLOG-FGG is displayed. For a description of the output, see the section following the operand description.

WRITE — All messages issued to message destination SYSLOG and still stored in buffer memory are written to the current SYSLOG file immediately.

This function is useful if the SYSLOG file, opened as a simple file, is to be evaluated in run mode. All messages generated by openUTM up to this time which have SYSLOG as their destination are covered by this evaluation.

However, to evaluate SYSLOG in run mode, it is better to open SYSLOG as an FGG. This enables you, before starting an evaluation, to switch over file generation with KDCSLOG SWITCH and to log all messages generated by openUTM up to this time. In other words, openUTM writes the message

buffer to the “old” SYSLOG file automatically before it switches over. However, the evaluation does not cover any of the messages generated after the switch time.

SIZE=fg_size

The KDCSLOG SIZE=fg_size command is only executed if SYSLOG was opened as an FGG.

fg_size redefines the control value for automatic size monitoring of the SYSLOG file. For *fg_size*, enter the desired control value representing a number of UTM pages (e.g. SIZE=100 defines a control value of 100 times the size of a UTM page).

fg_size ≥ 0 must be entered. If *fg_size* < 0 is entered, openUTM refuses to execute the command.
With fg_size=0 you can switch off automatic size monitoring. With *fg_size* > 0, automatic size monitoring is switched back on. Entries for *fg_size* between 1 and 99 are automatically replaced by 100. Values greater than 100 are accepted without changes as control values.

Minimum value: 100
Maximum value: (2^{31} -1)

SWITCH

Is only executed if SYSLOG was opened as an FGG.
KDCSLOG SWITCH prompts openUTM to switch the SYSLOG file to the next file generation.

openUTM guarantees that no more messages are written to the old SYSLOG file generation once this command has been executed successfully.

Before switching to a new file generation, openUTM continues writing messages stored in the internal message buffer to the old file generation.

B B B B B B B B

Please note the following points in UTM applications on BS2000 systems: Successful execution of a KDCSLOG SWITCH command by openUTM does not mean that you have immediate access to the new file generation. The old file generation can be kept open for an extended period of time by UTM processes, for example because the processing of a program unit started before the switch has not been completed and because no message with SYSLOG as its destination has yet been written by the relevant process.

B B B B

You can use KDCSLOG INFO as a query to find out which SYSLOG file generations have already been closed by all UTM processes, i.e. all file generations less than LOWEST-OPEN-GEN (see description of output from page 781).

SWITCH,SIZE=fg_size

Is only executed if the SYSLOG was opened as an FGG.

With KDCSLOG SWITCH,SIZE=*fg_size* you can switch the SYSLOG to a new file generation and, at the same time, redefine the control value for automatic size monitoring of the subsequent file generations. To this end, openUTM guarantees either that both actions are performed successfully, or that neither is. In other words, openUTM only sets the new control value if the SYSLOG switch operation was successful.

If openUTM is not able to switch to the next file generation, the control value does not change. Size monitoring is suspended and openUTM ignores the value specified for *fg_size*. Size monitoring cannot be reset until after a subsequent successful switch attempt (KDCSLOG SWITCH). If *fg_size* is not specified at this time, openUTM adopts the “old” value of *fg_size* as its control value.

For further information about the function, restrictions and possible values of *fg_size*, see the description of the operands SWITCH and SIZE=fg_size.

Output from KDCSLOG INFO:

```
SYSLOG FILE NAME               filename
FILE GENERATION GROUP          fgg
LAST SWITCH                    last-switch
SIZE CONTROL                   control

CURRENT SYSLOG SIZE            csp UTM PAGE(S)   =  csk KB
SIZE CONTROL VALUE             scp UTM PAGE(S)   =  sck KB
SYSLOG FILE                    rel% FILLED

FILE GENERATIONS OF APPL       START-GEN           start-gen
                               LOWEST-OPEN-GEN     low-gen
                               CURRENT-GEN         curr-gen

FILE GENERATIONS               BASE-GEN            basis-gen
                               FIRST-GEN           first-gen
                                  LAST-GEN               last-gen
```

Explanation of the output

SYSLOG FILE NAME
: Name of the current SYSLOG file. If the SYSLOG was opened as an FGG, the generation number of the current file generation is displayed with it.

FILE GENERATION GROUP
: Shows whether the SYSLOG was opened as an FGG or as a simple file.

  YES
  The SYSLOG was opened as an FGG.

  NO
  The SYSLOG was opened as a simple file.

LAST SWITCH
: Only output if the SYSLOG was opened as an FGG.
  LAST SWITCH indicates whether the last attempt by openUTM to switch to the next file generation executed without errors. The following values are possible:

  SUCCESSFUL
  The last switch attempt executed without errors.

  FAILED
  When openUTM last attempted to switch, an error occurred. openUTM was unable to switch to the next file generation.

  NONE
  No switch attempt was made in the current application.

SIZE CONTROL
: Only issued if the SYSLOG was opened as an FGG.
  SIZE CONTROL indicates whether the automatic size monitoring function is switched on. The following values are possible:

  ON
  Size monitoring is switched on

  OFF
  Size monitoring is switched off

  SUSPENDED
  The last attempt to switch to another file generation failed (for LAST SWITCH, the word FAILED is displayed). For this reason, size monitoring is suspended.
  Measure: you can try to switch SYSLOG again using KDCSLOG SWITCH. If the switch operation executes without errors, size monitoring is activated again automatically by openUTM.

CURRENT SYSLOG SIZE
: Present size of the SYSLOG file/current file generation; issued in number of UTM pages (*csp*) and in kilobytes (*csk*).

All the following information is only issued if the SYSLOG was opened as an FGG.

SIZE CONTROL VALUE
: Set size control value of the automatic size monitoring operation. The control value is issued in numbers of UTM pages (*scp*) and in kilobytes (*sck*). The kilobyte value is not displayed for very large control values (e.g. for 2^{31} KB).
If 0 is output as the SIZE CONTROL VALUE, size monitoring is switched off.

SYSLOG FILE .... % FILLED
: Is output if the automatic size monitoring function is switched on. The value indicates the percentage of the SYSLOG file already used up relative to the defined size control value (SIZE CONTROL VALUE). If size monitoring has been suspended by openUTM, the SYSLOG file can actually be filled by more than 100%. When this occurs, the message SYSLOG FILE “>100% FILLED” is output.

START-GEN
: Generation number of the first SYSLOG file generation written by openUTM in the current application run.

LOWEST-OPEN-GEN
: Generation number of the oldest SYSLOG file generation still kept open by a process in the application.

CURRENT-GEN
: Generation number of the file generation in which openUTM is currently keeping a log.

BASE-GEN
: Generation number of the defined basis of the SYSLOG FGG.

FIRST-GEN
: Generation number of the first valid file generation of the SYSLOG FGG.

B B
: On BS2000 systems, this is the same as the FIRST-GEN from the SHOW-FILE-ATTRIBUTES command.

LAST-GEN
: Generation number of the last valid file generation of SYSLOG-FGG.

B B
: On BS2000 systems, this is the same as the LAST-GEN from the SHOW-FILE-ATTRIBUTES command.

Output from KDCSLOG WRITE:

1. If openUTM is able to write the message buffer properly to the SYSLOG, openUTM issues the following message:

   ```
   **** SYSLOG BUFFER WRITTEN ****
   ```

2. If the message buffer for command processing is empty, the following message is issued:

   ```
   **** SYSLOG BUFFER IS EMPTY ****
   ```

3. If openUTM is not able to write the message buffer properly to the SYSLOG, the following message is issued:

   ```
   **** SYSLOG BUFFER NOT WRITTEN ****
   ```

Output from KDCSLOG SIZE=fg_size:

1. When *fg_size* ≥ 0 is displayed, if the SYSLOG for the application was opened as an FGG, then the following form of text is issued:

   ```
              NEW     OLD

      SIZE    100       0    COMMAND ACCEPTED – MINIMUM SIZE TAKEN
   ```

   The additional text COMMAND ACCEPTED- MINIMUM SIZE TAKEN is only issued if a value of between 1 and 99 is entered for *fg_size*.

2. If the SYSLOG is not opened as an FGG, the following message is issued:

   ```
   COMMAND REJECTED – SYSLOG FILE IS NO FGG
   ```

Output from KDCSLOG SWITCH:

1. If openUTM was able to switch the SYSLOG successfully, the following message is issued:

   ```
   *** SYSLOG SWITCH ACCEPTED ***
   ```

2. If the SYSLOG was not opened as an FGG, the following message is issued:

   ```
   *** SYSLOG SWITCH REJECTED – SYSLOG FILE IS NO FGG ***
   ```

3. If an error occurs during the switch operation, openUTM issues the following message:

   ```
   *** SYSLOG SWITCH REJECTED ***
   ```

KDCSWTCH - Change the assignment of clients and printers to LTERM partners

KDCSWTCH allows you to redefine the assignment of clients and printers (PTERM) to LTERM partners.

KDCSWTCH is only permitted in standalone UTM applications.

KDCSWTCH has the following effect:

– the existing assignment of a client/printer to an LTERM partner is cancelled and
– the client/printer is assigned to the specified LTERM partner.

This function can only be performed when no logical connection exists between the client/printer and the UTM application.

With KDCSWTCH you can, for example, assign another printer to a printer pool. In a printer pool, several physical printers are assigned to one LTERM partner.

If on the other hand you wish to assign an LTERM partner to a printer to which, in turn, a printer control LTERM is assigned (CTERM), then the control identification (CID) of that printer must be unique within the printer control LTERM range.

W
W

openUTM on Windows systems does not support any printers. Refer to the manual supplement file supplied with openUTM for any changes.

Restriction

Reassignment of the LTERM partner is possible only for terminals and printers. The assignment to an LTERM partner specified on configuration cannot be changed

- for UPIC clients
- for TS applications (APPLI/SOCKET) generated as interactive partners
- for clients that connect to the application via an LTERM pool
- for LTERMs that belong to an LTERM bundle or an LTERM group

If you assign a new LTERM partner to a terminal or printer, the LTERM partner may not be assigned to a terminal/printer of another protocol type (either currently or in the past). Distinctions are drawn here between the following 4 protocol types: terminals, TS applications, printers and RSO printers.

KDCSWTCH is therefore rejected if:

- the client specified in *ptermname* is a UPIC client or a TS dialog application (PTYPE=UPIC-R/L or PTYPE=APPLI/SOCKET) or
- the LTERM partner specified in *ltermname* was previously assigned to a UPIC client or a TS dialog application or if
- the LTERM partner specified in *ltermname* is assigned to an LTERM pool or if.
- B B the printer specified in *ptermname* is an RSO printer (PTYPE=RSO) and the LTERM partner specified in *ltermname* was previously assigned to a normal printer.
- the LTERM partner specified in *ltermname* belongs to an LTERM group or an LTERM bundle.

Period of validity of the change

These changes remain in force after the end of the application.

B
```
KDCSWTCH␣  ltermname,ptermname,proname [ ,applname ]
```
X/W
```
KDCSWTCH␣  ltermname,ptermname [,proname [ ,applname ] ]
```

For administration using message queuing you must enter KDCSWCHA.

ltermname Name of the LTERM partner to which the client or printer should be assigned. The LTERM partner must exist in the configuration of the UTM application.

ptermname, proname, applname
Uniquely identifies the client/printer.

ptermname
Name of the client or printer (PTERM name)

proname Name of the processor on which the client is running or to which the client or printer is connected.

B The *proname* entry is mandatory in UTM applications on BS2000 systems.

X/W X/W In UTM applications on Unix, Linux or Windows systems, *proname* only has to be entered if the client or printer is not connected locally.

X/W X/W Default value in openUTM on Unix, Linux or Windows systems:
Blanks for local clients/printer.

applname This entry is only meaningful for UPIC clients and TS applications.
For *applname*, please enter the name of the UTM application which was assigned to the client when it was entered in the configuration.

The *applname* entry is mandatory if the BCAMAPPL name assigned to the client does not match the name of the UTM application defined for KDCDEF generation in MAX APPLINAME. If *applname* is not entered, the command is rejected with this message:

```
BCAMAPPL–NAME ´applname´ INVALID OR NOT DEFINED
```

Default:
Name of the application defined in the KDCDEF control statement MAX in the APPLINAME operand.

Output from KDCSWTCH

The new and old assignments between client/printer and LTERM are displayed on the administrator terminal. The following section shows you the output for this call:

```
KDCSWTCH ltermname1,ptermname1,proname1,applname1
```

Here the LTERM partner *ltermname1* is assigned to the client *ptermname1*.

PTERM	PRONAM	BCAMAPPL	N E W LTERM		O L D LTERM	
			ltermname1	ltermname2	ltermname1	ltermname2
ptermname1	proname1	applname1	x			x
ptermname2	proname2	applname2			x	

Explanation of the output

openUTM outputs the old and new assignment for the client *ptermname* entered in the KDCSWTCH call (here *ptermname1*), and for the client assigned to the LTERM partner *ltermname1* before the KDCSWTCH call (here *ptermname2*).

Before the KDCSWTCH call, LTERM partner *ltermname2* was assigned to client *ptermname1* and client *ptermname2* was assigned to LTERM partner *ltermname1* (see column headed OLD LTERM). Both assignments are cancelled by the KDCSWTCH call.

If *ptermname1* and *ptermname2* are clients (i.e. not printers), then the old assignments linking LTERM partners to PTERMs are cancelled.

If *ptermname1* and *ptermname2* are printers, the old assignment of *ltermname1* to *ptermname2* is not cancelled. A printer pool is then always created at this point, i.e. both printers are assigned to the LTERM partner *ltermname1*.

PTERM Name of the client or printer

PRONAM Name of the processor on which the client/printer is located

BCAMAPPL Name of the local UTM application through which connection to the client/printer is established

NEW LTERM Name of the LTERM partner to which the client/printer was assigned by the KDCSWTCH call

ltermname1
Name of the LTERM partner which was assigned to the client/printer with the KDCSWTCH call

ltermname2
Name of the LTERM partner which was assigned to the client/printer before the KDCSWTCH call.

OLD LTERM Name of the LTERM partner to which the client/printer was previously assigned.

Example: Combining printers to form printer pools

Printers *pterm1* and *pterm2* are to be combined to form a printer pool. The LTERM partner of the printer pools is to be *lt-bundle*.

Assignment before the KDCSWTCH call:
Printer *pterm2* is already assigned to the LTERM partner *lt-bundle*. The printer *pterm1* is assigned to the LTERM partner *lt-print*.

Call:

```
KDCSWTCH lt-bundle,pterm1,proname1,applname1
```

Output:

PTERM	PRONAM	BCAMAPPL	N E W LTERM		O L D LTERM	
			lt-bundle	lt-print	lt-bundle	lt-print
pterm1	proname1	applname1	x			x
pterm2	proname2	applname2	x		x	

KDCTAC - Lock/release transaction codes and TAC queues

KDCTAC allows you to lock transaction codes and TAC queues and remove locks that were set during generation or by means of administration functions.
With the exception of the KDCTAC transaction code, this function can be applied to all transaction codes and TAC queues in the application.

Effect in UTM cluster applications

In UTM cluster applications, KDCTAC applies globally to the cluster.

Period of validity of the change

The event-driven service KDCMSGTC is locked only for the duration of the current application run.
For all other TACs the change remains in force beyond the end of the application.

```
KDCTAC␣    TAC={ tacname | (tacname_1,tacname_2,...,tacname_10) }

           ,STATUS={ OFF | HALT | KEEP | ON }
```

For administration using message queuing you must enter KDCTACA.

TAC=(tacname_1,...,tacname_10)
: Name of the transaction code or TAC queue to be administered. You can enter a maximum of 10 transaction codes or TAC queues per call. If you only enter one TAC name you do not need to enter the parentheses.

  The list must not contain the transaction code KDCTAC.

STATUS=

OFF
: Transaction codes or TAC queues *tacname_1,...,tacname_10* are to be locked.

  Transaction codes:
  With STATUS=OFF you can only lock service TACs, i.e. TACs configured with CALL=FIRST or CALL=BOTH. Locking with OFF causes UTM to stop accepting jobs for this TAC with immediate effect. If a TAC configured with CALL=BOTH is disabled, it can still be called as a follow-up TAC in a service.

  TAC queues:
  The TAC queues are locked for write access; read access is possible.

HALT
: Transaction codes or TAC queues *tacname_1,...,tacname_10* are to be locked completely.

Transaction codes:
A complete lock on a TAC means that, with immediate effect, no more program unit runs can be started for this TAC. This in turn means that no further jobs are accepted for the TAC and, over and above this, it is disabled as a follow-up TAC in an asynchronous or dialog service.

If a completely disabled TAC is called as a follow-up TAC, the service is terminated with PEND ER (74Z). Asynchronous jobs already queued in the TAC message queues are not started. They remain in the message queue until the TAC is released again (STATUS=ON) or set to STATUS=OFF.

TAC queues:
The TAC queues are locked for read and write access.

KEEP May only be specified for TAC queues and asynchronous transaction codes that are also service TACs (CALL=FIRST/BOTH).

Transaction codes:
The transaction code is disabled. Jobs for this transaction code are accepted, but they are not processed. The jobs are simply placed in the job queue for the transaction code. They are not processed until you change the transaction code status to ON.

You can use the status KEEP to collect jobs that are to be processed at a later time at which the degree of utilization of the application is lower (e.g. at night).

In order to avoid an overload of the page pool due to too many jobs being temporarily stored, you should limit the size of the job queue of the transaction code. For this you must set the parameter QLEV appropriately when you generate the transaction code.

TAC queues:
The TAC queues are locked for read access; write access is possible.

ON The transaction codes or TAC queues *tacname_1,...,tacname_10* are released. Any locks set during generation or by means of administration functions are cancelled.

Output from KDCTAC

The new and old properties of the transaction codes are output to the administrator terminal.

TAC	STATUS	
	NEW	OLD
tacname_1	ON| OFF| HLT	ON| OFF| HLT
tacname_2	.	.
.	.	.
.	.	.
.	.	.

KDCTCL - Change number of processes of a TAC class

It only makes sense to call the command KDCTCL, if jobs are processed in your application using a “limited number of processes for TAC classes”, i.e. no TAC-PRIORITIES statement is generated (see the openUTM manual “Generating Applications”).

With KDCTCL you can:

- get information on the current settings for the TAC classes. To do this, enter KDCTCL without the operands TASKS and TASKSFREE.
- change the maximum number of processes that can process TACs of a TAC class at the same time. You can only change this value if the KDCDEF generation of your application does not contain the TAC-PRIORTIES statement.

  The number of processes that you can allow for individual TAC classes is limited by the maximum number of processes defined in the MAX statement during the KDCDEF generation (operands TASKS, ASYNTASKS and TASKS-IN-PGWT).
  If you enter a higher number of processes, KDCTCL is rejected.

  After the KDCTCL, the actual number of processes set for processing TACs of a TAC class may be smaller than the value set with KDCTCL. The actual number of processes depends on the current number of processes for the entire application (set with the start parameter TASKS or by administration, e.g. using KDCAPPL).

You can define the maximum number of processes for a TAC class in one of two ways: either by entering the number of processes allowed to process TACs in one TAC class at the same time (TASKS operand); or by entering the minimum number of processes in the application that are to be kept available for processing the TACs in other TAC classes (TASKSFREE operand). The following section explains the difference between TASKS and TASKSFREE:

- When you use TASKS, the maximum number of processes available to the specified TAC class is independent of the number of processes currently available for the entire application program. This means that the number of processes in the TAC class remains constant even if the number of processes in the entire application is reduced. This applies until such time as the number of processes in the TAC class and the number in the entire application are identical.

  The use of the TASKS operand can (in extreme cases) cause processes in one TAC class to hinder those in all other TAC classes.
- When you use TASKSFREE, the maximum number of processes available to the specified TAC class depends, in the dynamic context, on the number of processes currently available for the entire application program. The reserve number specified in TASKSFREE is always kept free for processes in other TAC classes.

The maximum number of processes for one TAC class is then obtained in the following manner:

– Dialog TAC classes (1 - 8): current number of all processes available for dialog TACs in the entire application program (TASKS), less the number in TASKSFREE, but at least one process
– Asynchronous TAC classes (9 -16): current number of all processes available for asynchronous TACs in the entire application program (TASKS), less the number in TASKSFREE.

Effect in UTM cluster applications

In UTM cluster applications, KDCTCL applies locally in the node.

Period of validity of the change

This change does not remain in force beyond the end of the application. The available number of processes is determined by the most recent KDCTCL call entered.

```
KDCTCL␣    CLASS=tacclass
           [ ,{ TASKS=number_tasks | TASKSFREE=number_tasks } ]
```

For administration using message queuing you must enter KDCTCLA.

CLASS=tacclass

Number of the TAC class for which the number of processes should be changed. For *tacclass* you can enter a number between 1 and 16 ($1 \leq tacclass \leq 16$).

TASKS=number_tasks

May only be specified if no priority control is generated for the application, i.e. if the application is generated without TAC-PRIORITIES.

Specifies how many processes in the application are allowed to process TACs in TAC class *tacclass* at the same time.
With TASKS you define the absolute number of processes for a TAC class.

Minimum value of *number_tasks*:
For dialog TACs (class 1-8) *number_tasks* must be ≥ 1. Otherwise dialog services would be locked and the users on the terminal would have to wait until services were released again.

For asynchronous TACs (class 9-16), *number_tasks* must be =0.

Maximum value of *number_tasks*:
The permitted maximum value for *number_tasks* depends on the following factors:

– On whether the TAC class was generated with PGWT=YES or with PGWT=NO.
PGWT=YES means that the program units in the TAC class can run with lock calls (e.g. KDCS call PGWT).

– On the values for TASKS, TASKS-IN-PGWT and ASYNTASKS generated statically in the KDCDEF control statement MAX.

See the following table for the permitted value ranges for TASKS.

TAC class	PGWT=	Permitted maximum value
1 - 8 (dialog TACs)	NO	TASKS *)
	YES	TASKS-IN-PGWT *)
9 - 16 (asynchronous TACs)	NO	ASYNTASKS *)
	YES	The smaller of the values: ASYNTASKS and *) TASKS-IN-PGWT *)

*) As generated statically in the KDCDEF control statement MAX

TASKSFREE=number_tasks

May only be specified if no priority control is generated for the application, i.e. if the application is generated without TAC-PRIORITIES.

In TASKSFREE you specify how many processes of the application are to be reserved for processing other TAC classes than the one specified.

If *number_tasks* is greater than the number of processes available to the entire application program, the following occurs:

– if *tacclass* is a dialog TAC class, one process remains available for processing its TACs;
– if *tacclass* is an asynchronous TAC class, the number of processes available to it = 0.

Minimum value of *number_tasks*: 0

Maximum value of *number_tasks*:
The permitted maximum value for *number_tasks* depends on the statically generated values for TASKS and ASYNTASKS in the KDCDEF control statement MAX.

See the following table for the permitted value ranges for TASKSFREE.

TAC class	PGWT=	Permitted maximum value
1 - 8 (dialog TACs)	NO	TASKS - 1 *)
	YES	TASKS - 1 *)
9 - 16 (asynchronous TACs)	NO	ASYNTASKS *)
	YES	ASYNTASKS *)

*) As statically generated in the KDCDEF control statement MAX

Output from KDCTCL

If you enter KDCTCL without TASKS or TASKSFREE, you are only shown the currently set values. Otherwise, the output for the specified TAC class shows you the new and old process numbers. Output is displayed on the administrator terminal.

```
TACCLASS      TASKS        TASKS-FREE

             NEW    OLD      NEW    OLD

tac-class    number number   number number
```

Explanation of the output

TACCLASS Number of the TAC class

TASKS Absolute number of processes available for processing the TACs in this TAC class. If you called KDCTCL ... TASKSFREE=, the following value is displayed:

Process number currently set for the application - TASKFREE

TASKS-FREE Number of processes kept free for other TAC classes. If you entered KDCTCL ... TASKS=, the output of TASKS-FREE is always 0 to show that you made an absolute entry for this TAC class.

Example

The following table illustrates the impact of various changes to the number of processes:

	Dialog TACs			Asynchronous TACs		
Action	**CURRENT TASKS**	**TASKS FREE**	**TASKS**	**CURRENT TASKS**	**TASKS FREE**	**TASKS**
Initial status	4	0	3	3	0	3
Change TASKS-FREE 0 → 2	4	2	2	3	2	1
Change CURRENT TASKS reduced by 2	2	2	1	1	2	0

CURRENT-TASKS
This represents the maximum number of processes that can currently be used at the same time for the application (dialog TACs) or
the maximum number of processes that can currently process asynchronous jobs at the same time (asynchronous TACs).

TASKS
Designates the appropriate maximum number of processes for the specified TAC class.

TASKS-FREE
Designates the number of processes reserved for the other TAC class.

KDCUSER - Change user properties

With KDCUSER you can:

- disable or release user IDs for the application
- define, change or delete passwords for user IDs.

Effect in UTM cluster applications

In UTM cluster applications, KDCUSER applies globally to the cluster.

Period of validity of the change

Changes remain valid beyond the end of the application.

```
KDCUSER␣   USER={ username | (username_1,username_2,...,username_10) }
           [ ,PASS=password ]
           [ ,STATUS={ ON| OFF } ]
```

For administration using message queuing you must enter KDCUSERA.

USER=(user1,user2,...)
: Names of the user IDs to be administered. You can enter a maximum of 10 names per call. If you only enter one name you do not need to key in the parentheses.

PASS=password
: Issue, change or delete password for the user ID.

  The password can be up to 8 characters in length. If the specified password is shorter than 8 characters openUTM fills the balance with blanks.

  You can enter the password as a hexadecimal string (16 half bytes) in the form X´.....´ or as a character string C´....´.

  Example:
  Hexadecimal string: X´F1F2F3F4F5F6´
  Character string: ´ABCDEFGH´

  You delete a password by entering PASS=C´␣´ (blank). If you enter 8 binary zero characters (X´0000000000000000´) you will not change the password.

  You can only delete the password if

  – the minimum length defined for the password when the user ID is entered is 0
  – no complexity level is defined for the user ID (NONE).

If a password with a restricted period of validity is generated for a user ID, you cannot enter the old password as the new password when changing the password.
If the application has been generated with SIGNON GRACE=NO, the generated period of validity from the time of the change also applies to the new password.
If a password with a restricted period of validity is deleted, no period of validity applies. If a new password is issued after this, the period of validity is restored.

STATUS=

ON Releases the user ID

OFF Disables the user ID. This lock takes effect when the user next attempts to sign on. This function does not work for the administrator.

Output from KDCUSER

The old and the new status of the administered user IDs are displayed at the administrator terminal along with an indication of the password having been changed, where applicable.

```
 USER            STATUS
                NEW         OLD
user1      ON | OFF    ON | OFF          PASSWORD CHANGED
   .
   .
   .
```

13 Administering message queues and controlling printers

There are two ways to administer message queues and control printer outputs:

1. using the KDCS program interface with the DADM (delayed free message **adm**inistration) and PADM (printer **adm**inistration) functions
2. using WinAdmin or WebAdmin, which provides you with DADM and PADM functionality in a graphical user interface

The following sections describe how to use the DADM and PADM functions. The requirements and conditions specified here also apply to administration with WinAdmin or WebAdmin.

DADM enables you to administer jobs and messages buffered in local message queues and waiting for processing. With the exception of the dead letter queue, the message queues in openUTM are recipient-specific, i.e. all asynchronous jobs in any one queue are intended for the same recipient. Recipients can, for example, be: asynchronous TACs in your own or in a remote application (background jobs are located in these queues), LTERM partners for terminals, TS applications or printers (output jobs are buffered in these queues), user IDs and temporary queues. The dead letter queue is a TAC queue containing messages to various recipients that have not been processed correctly. For further information on message queues, see the section “Administering message queues (DADM)” on page 805 and the detailed information in the openUTM manual “Concepts und Functions”.

With PADM you can control the output of asynchronous messages to printers, i.e. you can influence print output and administer the printers yourself. In order to administer a printer and control print output using PADM functions, the printer must be assigned to a printer control LTERM (see page 802).

You can use DADM to execute the following functions:

- to output information about jobs and messages in buffer storage in a message queue
- to prioritize a job or message in a queue to ensure that it is processed before all other asynchronous jobs in the queue
- to cancel a job or message, i.e. to delete it from the queue.
- Move messages from the dead letter queue in order to process them.

PADM allows a program unit to execute the following functions to control printer output:

- switch a special confirmation mode on and off which entails confirming every print job before the next output job can be processed.
  In UTM cluster applications, this action applies globally to the cluster.
- repeat print jobs, e.g. after a successful sample printout. For this, the confirmation mode has to be switched on.
  In UTM cluster applications, this action applies locally in the node.
- output a list of print jobs which still have to be confirmed.
  In UTM cluster applications, this action applies locally in the node.

With PADM, a program unit can also execute the following print administration functions:

- disable and re-enable a printer.
  In UTM cluster applications, this action applies globally to the cluster.
- establish or shut down a connection to a printer.
  In UTM cluster applications, this action applies locally in the node.
- change the assignment of printers to LTERM partners, e.g. if one printer fails, the LTERM partner of this printer and the attached message queue can be assigned to another printer which then processes the print jobs waiting in this queue.
  This function is only permitted in standalone UTM applications.
- group printers into pools. To do this, you assign several printers to one LTERM partner. The message queue of the LTERM partner is then processed jointly by all the printers in the pool. For further information about printer pools, also see openUTM manual “Generating Applications”.
  This function is only permitted in standalone UTM applications; in UTM cluster applications, printer pools can only be generated statically.
- output information to a printer.

UTM administration privileges are not always mandatory for administering printers with PADM calls. Refer to “Authorizations concept (BS2000, Unix and Linux systems)” on page 802 for an explanation of the authorization level you require to start program units with DADM and PADM calls.

The sample program units KDCDADM and KDCPADM are supplied with openUTM: these units use the functions of DADM and PADM. You can use these program units to administer asynchronous jobs and to control print jobs and printers without having to write program units yourself. The following description uses to refer to the corresponding functions of KDCDADM and KDCPADM.

If you create your own program units using PADM and DADM, you have the option of designing your own user interface in the program unit, e.g. data input using formats on BS2000 systems.

The DADM and PADM calls are described in the openUTM manual „Programming Applications with KDCS”. The KDCDADM and KDCPADM program units are described in the section “UTM program units for DADM and PADM functions” on page 820.

Before you can use sample programs KDCDADM and KDCPADM or your own program units with PADM or DADM calls, you must first record the program units in the configuration of the application, either statically or dynamically, and assign transaction codes to them.

W
W
W
W

openUTM on Windows systems does not support printers. The KDCS call PADM and the program unit KDCPADM are available, however, they are irrelevant for UTM applications running on Windows systems. Administration privileges are required for actions taken using DADM or KDCDACDM.

13.1 Authorizations concept (BS2000, Unix and Linux systems)

B/X PADM and DADM are not functions of the program interface for administration. For this reason, services which use PADM and DADM have a different authorizations concept. This authorizations concept enables them to administer their own output jobs to the “local” printers without administration privileges. Users can also perform administration of the “local” printers without any special privileges.

B/X To do this, you must create printer control LTERMs for the printers and assign them to the printers that are to be administered “locally”, i.e. by a user/client without administration privileges. The related printers and their queues can then be administered by every user or client who signs on using the printer control LTERM.

B/X Administration privileges are required for the following administration tasks:

- B/X Administration of background jobs and output jobs for terminals or remote TS applications.
- B/X Administration of output jobs and printers using any LTERM partner.
  A user who has UTM administration privileges can administer all printers on all printer control LTERMs and all asynchronous jobs irrespective of which LTERM partner was used to initiate the services.
- B/X Administration of service-controlled queues (USER, TAC and temporary queues).

B/X **Printer control LTERM - administration of “local printers”**

B/X A printer control LTERM is an LTERM partner that is entered as a dialog partner (*usage*=D). A client or a terminal user can log into an application via this LTERM partner. From the terminal or client, the printers and queues assigned to the printer control LTERM can be administered.

B/X Printers are assigned to the printer control LTERM as follows:
An LTERM partner configured as an output medium is assigned to each printer (*usage*=’O’). openUTM “sends” all output jobs for this printer to the LTERM partner of that printer, i.e. openUTM writes the output job to the message queue of the LTERM partner - the queue for the printer concerned. You can also assign several printers (a printer pool) to an LTERM partner. All these printers then work with the same queue.

B/X You assign the LTERM printer partners to the printer control LTERM.

B/X To do this, when creating the LTERM partners in CTERM/*kc_lterm_str.cterm* (CTERM=**C**ontrol **TERM**inal) you must specify the printer control LTERM to which the relevant printer is assigned. In CTERM/*kc_lterm_str.cterm* you enter the name of the printer control LTERM (name of the LTERM partners).

B/X You can assign individual printers and even printer pools to one printer control LTERM. You must define a printer r ID for each printer to which a printer control LTERM is assigned. This printer ID must be unique in the printer control LTERM range because the printer control LTERM uses this printer ID to address the printers directly. You must pay particular attention to the unique nature of each printer ID in printer pools. You must also define a separate printer ID for each of the printers in the pool. These printer IDs must be assigned to the correct printers when they are entered in the configuration.

B/X The figure on page 804 provides an example of a configuration with KDCDEF.

B/X In order to restrict access to the printer control LTERM to a defined number of people, you can assign a lock code to the printer control LTERM. Similarly, you can also protect the PADM and DADM program units by means of lock codes or access lists. This enables you to define which administration functions can be performed by users/clients. In any event, you should assign all keycodes for the print administration and printer control program units to the printer control LTERM (for details of the lock code/keycode concept, see the openUTM manual “Concepts und Functions”).

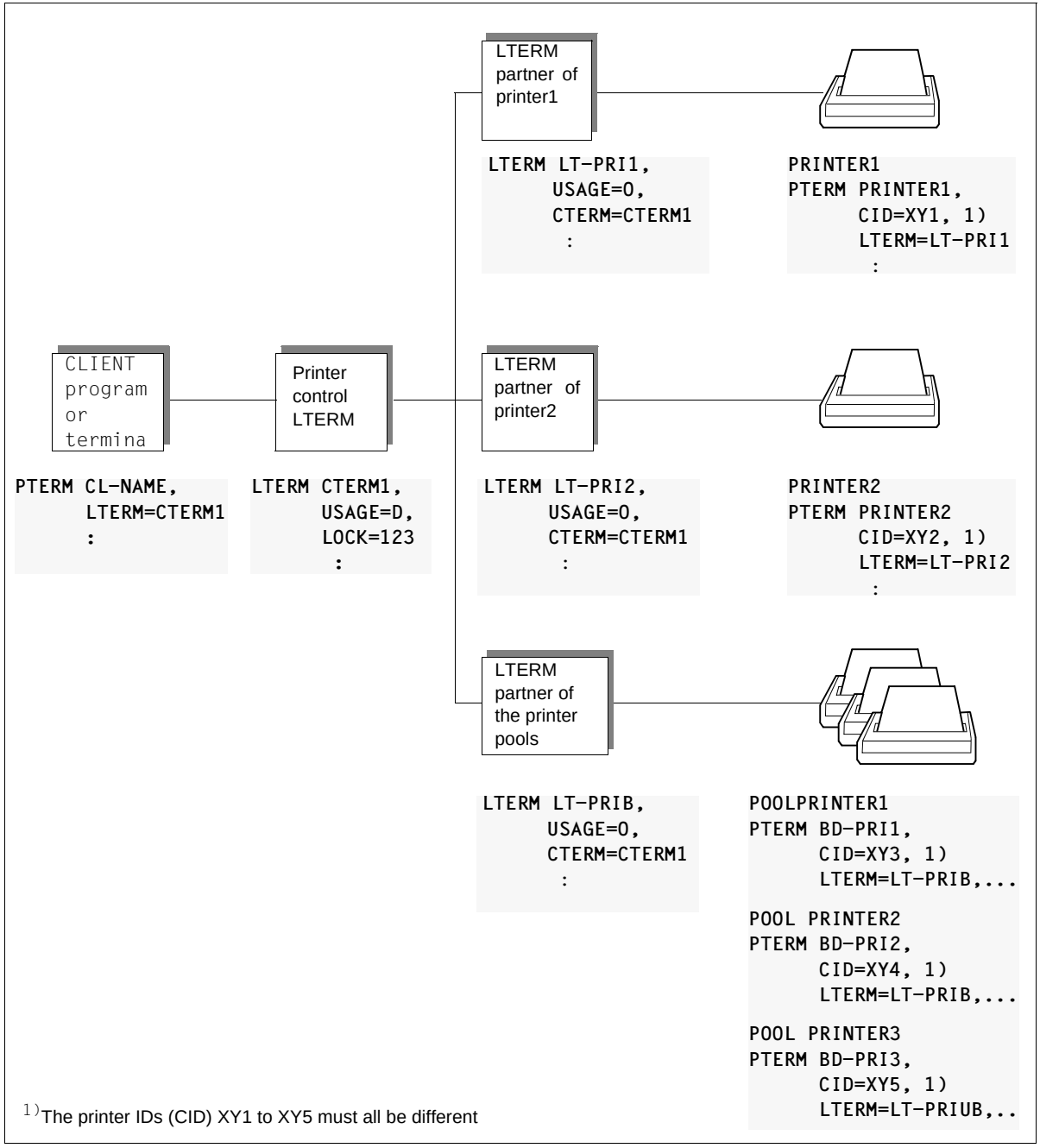
B/X A user/client can start services via a printer control LTERM which:

B/X
- administer associated printers by means of PADM calls

B/X
- administer output jobs sent to the printer (DADM calls)

B/X
- control print jobs on these printers.

B/X You will need to write program units which use the DADM and PADM functions and which should be started from a printer control LTERM as dialog programs and you must assign dialog TACs to them.



Entering a printer control LTERM and the associated printers

13.2 Administering message queues (DADM)

You can use DADM to administer two different types of message queues. These are:

- UTM-controlled queues

  The asynchronous jobs created by a program unit are delivered to the recipient at the specified time. For messages to TACs, the associated program unit is started by openUTM.

- Service-controlled queues

  In these queues, processing is controlled not by UTM but by the program unit itself.

  There are three types of service-controlled queues available:

  1. USER queues

     A permanent queue is available to every user of an openUTM application under the user's user ID. The queue is accessible via the user ID. USER queues offer you the opportunity to send asynchronous messages to a UPIC user, for example.

  2. TAC queues

     Permanent queues with fixed names are created by generating TACs of the type ´Q´. In this way, queues can be implemented in remote UTM applications, for example, that are addressed by the local UTM application by means of an LTAC name.

     The dead letter queue KDCDLETQ is a TAC queue that is always available for backing up messages which could not be processed.

  3. Temporary queues

     Temporary queues can be created and deleted dynamically. The name of one of these queues can be created by the program unit or implicitly by openUTM. Temporary queues permit communication between two services, for example: A service sets up the queue and sends a message to the queue; another service reads the message and then deletes the queue.

     The maximum possible number of temporary queues is specified with the generation statement QUEUE.

     The KDCS calls QCRE and QREL are available to you to create and delete temporary queues. These calls are described in the openUTM manual „Programming Applications with KDCS".

You can administer messages in a queue using DADM at the KDCS program interface. FPUT and DPUT allow you to create background jobs, output jobs and messages for service-controlled queues. The actual function performed in each case by DADM depends on the operation modifier which you pass to UTM in the *kcom* field of the parameter area. The following operation modifiers are available:

- DADM RQ (**r**ead **q**ueue) for reading information about the messages in a message queue
- DADM UI (**u**ser **i**nformation) for reading user information about a message. User information is written by the job submitter and passed to the specified reception area when the message is created.
- DADM CS (**c**hange **s**equence) changes the sequence of messages in a queue. This function enables you to move a message from any position in the queue to the front of the queue. This message is then processed before any of the other messages in the queue.
- DADM DL (**d**e**l**ete) and DADM DA (**d**elete **a**ll) for deleting an individual message or all messages in a queue.

  When deleting job complexes with DADM DL, you can activate negative confirmation jobs. A job complex is an asynchronous job with a positive and/or negative confirmation job (see openUTM manual „Programming Applications with KDCS”, the MCOM call).

  When you delete messages using DADM DA, the messages are deleted with the following messages. A delete call like this is only executed:

  – in the case of UTM-controlled queues, when there is no job being processed for the specified destination

  – in the case of service-controlled queues, when no messages are currently being read

- DADM MV (**M**o**v**e) and DADM MA (**M**ove **a**ll) for moving one or all of the messages stored in the dead letter queue. The messages can be assigned to their original message queues, to any asynchronous TAC or to any TAC queue.

To enable openUTM to process a DADM message, you must uniquely identify the message queue and the message in the queue.

Identifying the message queue

The message queues in openUTM are recipient-specific, i.e. either openUTM or the program unit itself administers a separate message queue for each recipient of jobs or messages. A UTM-controlled message queue to be administered is uniquely identified when you specify the name of the recipient when making a DADM call. In the case of UTM-controlled queues, you specify, for example:

- in the case of output jobs, the name of the LTERM partner to which the terminal, the printer or the TS application is assigned
- in the case of background jobs, the name of the asynchronous TAC to which the job is directed

In the case of service-controlled queues, the name and the type of the queue are required for the purpose of identification.

You pass the name of the recipient for DADM RQ/DL/DA in the *kclt* field of the KB parameter area and the type in the *kcqtyp* field.

Identifying messages in a message queue

For every message, openUTM establishes a separate identification, also known as a job ID or DPUT-ID. This enables you to administer each message individually.

After a message has been processed by the recipient, or after a message has been deleted by the administration function, the job ID is released and can immediately be reassigned to another message by UTM. For this reason, in the case of DADM UI/CS/DL calls requesting unique identification of the message to be administered, it is also necessary to enter the time the message was created. This is the only way of preventing the wrong message from being cancelled by DADM DL.

In the case of DADM calls, you must pass a job ID and the time the message was created in the KB parameter area. You can determine both items of data using DADM RQ and use them in subsequent DADM calls.

If the messages (FPUT and DPUT messages) buffered in the KDCFILE are transferred to a new KDCFILE with the UTM tool KDCUPD, they are then assigned **new** job IDs.

13.2.1 Displaying information on messages in a queue - DADM RQ

With the DADM RQ call, openUTM supplies information about the messages in a queue. For every message, openUTM provides the job ID, the user ID of the job submitter, the origination time of the message and, in the case of time-controlled messages (DPUT messages), the earliest execution time. It also informs you whether a positive or negative confirmation job exists.

For a DADM RQ call, you enter the name of the recipient in the *kclt* field of the KB parameter area whose message queue is to be read. In the case of service-controlled queues, the type is also required in the *kcqtyp* field.

You can also output information about all the messages in a message queue or restrict the information output to just one message in the queue.

In the *kcrn* field of the parameter area, you enter the job ID of the message for which openUTM is to provide information. If you write any blanks in *kcrn*, openUTM informs you about the first message in the message queue for the recipient *kclt*.

The procedure for reading information about all messages in a message queue is as follows:

- In the first DADM RQ for a recipient, instead of a job ID you enter blanks in the *kcrn* field of the parameter area.
- UTM returns information about the first message in the message queue of the recipient. If at least one other message exists for the same recipient, openUTM writes the job ID of the next message in the queue to the *kcrmf* field of the KB return area.
- You call DADM RQ once again and write the job ID which openUTM returned in the *kcrmf* field to the *kcrn* field of the KB parameter area.
- UTM provides information about the second message and returns the job ID of the next message in the queue if another message exists.

This means that the message queue can be processed sequentially. When the information about the last message in the queue is read, UTM returns blanks to the *kcrmf* field.

A data structure exists for information returned from DADM RQ which you can place over the message area. The C data structure is called *kc_dadm* and is part of the header file *kcdad.h*. The corresponding COBOL data structure is called KCDADC.

See KDCDADM INFORM on page 824.

13.2.2 Reading user information about a message - DADM UI

In many cases, the information about messages provided by openUTM (see DADM RQ on page 808) is not sufficient to enable the administrator to uniquely identify a message. For this reason, the job submitter can store additional information when creating a message using the DPUT call: this information is known as *user information*. User information is written with the DPUT NI call or, in the case of confirmation jobs in job complexes, with DPUT +I or DPUT -I (see the openUTM manual „Programming Applications with KDCS").

This user information is not passed to the recipient of the message. However, it is linked to the job ID of the message and can only be read with DADM UI.

When a call is received from DADM UI, you must pass the job ID and the time the message was created in the KB parameter area. Both items of data can be determined in advance with DADM RQ.

You cannot read the user information relating to confirmation jobs in job complexes until the confirmation job has been activated.

See KDCDADM INFORM, LIST=LONG on page 824.

13.2.3 Prioritizing messages in the queue - DADM CS

The DADM CS call is advisable if, at a given point in time, several messages are in the processing queue for the same recipient. Using DADM CS, the specified message, identified by its job ID and the time it was created, is moved to first position in the message queue. You can determine the job ID and the time of the message's creation using DADM RQ.

Please note that you can only prioritize time-driven messages if the "earliest execution time" specified by DPUT at the time of the message's creation has already elapsed. Otherwise, UTM rejects the DADM-CS call (return code 40Z).

See KDCDADM NEXT on page 829.

13.2.4 Deleting messages from a queue - DADM DA/DL

DADM DA allows you to delete all messages to a given recipient which have not been processed at the time of the DADM DA call. In the case of service-controlled queues, messages that are currently being read cannot be deleted. If a service-controlled queue is deleted dynamically (KC_DELETE_OBJECT or QREL RL), the messages in this queue are lost. Messages already processed by the recipient are not deleted. With DADM DA calls you have to specify the name of the recipient in the *kclt* field of the KB parameter area.

With DADM DL you delete one specific message. To identify this message you have to enter the job ID and the time when the message was created. Both these items of data can be determined using DADM RQ.

If the specified message has already been processed by the recipient, the DADM DL call is rejected by openUTM (return code 40Z).

In particular, you cannot use DADM DA/DL to delete any print output which has already been started. To do this, you must follow the procedure described below:

1. Terminate the connection to the printer on which the job is being processed (PADM CS). openUTM also terminates the link to the printer if you disable the printer using PADM CS.
2. Delete the print job (DADM DL).
3. Restore the connection to the printer (PADM CS; see page 814).

If confirmation jobs are assigned to the message being deleted (DPUT jobs in job complexes) you can specify with DADM DL whether the negative confirmation job is to be activated when the message is deleted or whether the confirmation jobs are to be deleted together with the main job (*kcmod* field in the KB parameter area).

For information about job complexes and confirmation jobs, see the openUTM manual „Programming Applications with KDCS”.

See KDCDADM DELETE on page 822.

13.2.5 Move messages from the dead letter queue - DADM MA/MV

The dead letter queue is made up of messages which could not be processed and which have not been redelivered. In order to process these messages after any errors have been corrected, they must be assigned either to their original destination or to a new destination.

DADM MA allows you to move all messages stored in the dead letter queue. The messages can be assigned to their original message queues, to any asynchronous TAC or to any TAC queue.

DADM MV allows you to move a single message from the dead letter queue. You must specify the job ID and the time at which the message was generated in order to identify the message.

To identify the destination, specify:

- the TAC if the recipient of the messages is to be an asynchronous program,
- the name of the TAC queue if the recipient of the messages is to be a service-controlled queue,
- blanks if the messages are to be assigned to their original destination again.

If DADM MA is specified with KCLT=blank, messages whose destination no longer exists remain in the dead letter queue. You can assign these messages to asynchronous transaction codes or TAC queues as new destinations.

When moving messages from the dead letter queue, any QLEV that is defined and the STATUS of the recipient queue are ignored. This means that when moving messages, it is possible for the queue level to be exceeded and for messages to be sent to locked TACs.

The original destination of a message in the dead letter queue is available from the return information of the DADM RQ call.

KDCDADM MOVE on page 827

13.3 Administering printers and control print output (PADM)

You can use the KDCS call PADM to create program units which control the output of asynchronous messages on the printer and which administer printers. PADM functions are only able to administer printers which are assigned to a printer control LTERM.

Identifying printers during administration with PADM calls

Program units which are to control print output and administer printers must identify the printers uniquely. In order to be independent of the printer name, you must define a printer ID for every printer assigned to a printer control LTERM. The printer IDs are defined when the printers are entered in the configuration. The printer IDs must be unique in the printer control LTERM range.

A printer is rendered uniquely identifiable throughout an application by the name of the printer control LTERM to which it belongs and by virtue of its printer ID. Viewed in terms of the administration performed by the printer control LTERM, the printer ID provides adequate identification of the printer, e.g. when confirming printer output.

If you wish to control the print output of a printer, you do not need to send the printer ID of that printer to the program unit. It can be determined within a program unit with the help of PADM AI/PI calls.

13.3.1 Administering printers with PADM

openUTM provides the PADM call for printer administration functions. The actual function executed by PADM depends on the operation modifier which you pass to openUTM in the *kcom* field of the parameter area. The following operation modifiers are available:

- PADM PI (printer information) to read information about the printers assigned to a printer control LTERM
- PADM CA (change address) to assign a printer to a different LTERM partner
- PADM CS (change state) to change the printer status, i.e. disabling and re-enabling a printer, terminating or re-establishing a connection to a printer.

13.3.1.1 Querying information about a printer PADM PI

The call PADM PI returns the following information about every printer in a printer control LTERM (list not exhaustive):

– printer ID of the printer
– name of the related LTERM partner
– status of the printer, i.e. openUTM informs you whether or not the printer is currently disabled and if it is connected to the application or not
– number of print jobs in the printer queue
– number of time-driven print jobs in the printer queue and their earliest output time

You can output this information, e.g. to the printer control LTERM.

You can output information about a specific printer. To do this, you must enter its printer ID in the *kcrn* field of the parameter area. If you enter blanks in *kcrn*, openUTM informs you about the first printer.

You can also output information about all printers belonging to a printer control LTERM using the following procedure:

- At the first PADM PI, enter blanks in the *kcrn* field of the parameter area in order to read the information about the first printer.
- openUTM returns various items of information including the printer ID of the first printer. If at least one more printer is associated with this printer control LTERM, UTM writes the printer ID of the next printer in the *kcrmf* field of the KB return area.
- Call PADM PI again and write the printer ID which openUTM previously returned to *kcrmf* in *kcrn* of the KB parameter area.
- openUTM supplies information to the second printer and returns the printer ID of the next printer, provided that another printer exists etc.

When reading the information for the last printer, openUTM returns blanks to the *kcrmf* field.

A data structure exists for the information returned by PADM PI which you can place over the message area. The C data structure is called *kc_padm* and is part of the header file *kcpad.h:* the COBOL data structure is called KCPADC.

See KDCPADM INFORM, LIST=PRINTERS on page 831.

13.3.1.2 Changing the printer status - PADM CS

Using the PADM CS call you can perform the actions described in the following list. You define which action is to be performed in the *kcact* field of the parameter area.

- Disable a printer (*kcact*=OFF) or re-enable a printer which was previously disabled (*kcact*=ON).

  In UTM cluster applications, both actions apply globally to the cluster.

- Establish (*kcact*=CON) or terminate (*kcact*=DIS) a connection to a printer.

Output jobs to printers are always written to the message queue of the associated LTERM partner. If the printer is disabled or not connected, the data is buffered until the printer is re-enabled or the connection is re-established, or until you assign the LTERM partner to another printer, one which is not disabled, and connect this one to the system.

When a printer is disabled, the connection to it is established automatically and must be re-established explicitly after it has been released.

You cannot establish a connection to a disabled printer. To reconnect a disabled printer, proceed as follows:

1. Re-enable the printer. To do this, call PADM CS with *kcact*=ON.
2. Use PADM PI to confirm that openUTM has re-enabled the printer.
3. Call PADM CS with *kcact*=CON to re-establish the connection.

The first PADM call must not be performed in the same transaction as the other two.

See KDCPADM STATE on page 837.

13.3.1.3 Assigning a printer to another LTERM partner - PADM CA

You can use PADM CA to change the assignment of printers to LTERM partners. Enter the name of the new LTERM partner in the *kcadrlt* field of the parameter area. The new LTERM partner to which the printer is to be assigned must already feature in the configuration of the application, and the printer connection must be defined in it (*usage*=O). A printer can already be assigned to the LTERM partner. This old assignment is not cancelled.

This function is only permitted in standalone UTM applications

With this function you can generate printer pools during the application run by assigning several printers to one LTERM partner. All printers in the printer pool then process the queue for the LTERM partner.

However, if one printer fails, you can assign the LTERM partner for that printer to another printer together with the failed printer’s message queue. The new printer then processes the output jobs.

If a procedure which changes the assignment is started by a user or client on an LTERM printer control unit not authorized by administration then not only the LTERM partner but also the printer must lie in the responsibility area of the printer control LTERM. In other words, this printer control LTERM must be assigned to the LTERM partner, and the printer must previously have had an LTERM partner assigned to this printer control LTERM.

PADM CA is only permitted if the printer is not connected to the application. You can check this in advance using the PADM PI call. Owing to the fact that PADM CA is subject to transaction management, i.e. the fact that it is not executed until the end of a transaction, a connection to a printer may have been established from another service in the intervening period. For this reason, in a follow-up transaction, you should use PADM PI to check whether the action has indeed been executed.

See KDCPADM SWITCH on page 838.

13.3.2 Print control with PADM

Usually, print jobs are issued “without” print control, i.e. it is UTM that controls the output of messages to printers. Print output takes place in automatic mode in such cases. Automatic mode is set after the first time the application is started.

Print output “with” print control means that it is the user who has to control the output of messages. Print control can be performed in the following ways:

- procedures with PADM calls which are started by a printer control LTERM
- procedures with PADM calls which run under UTM administration privileges, e.g. the event service MSGTAC.

In UTM cluster applications, the change of mode (with/without print control) applies globally to the cluster.

openUTM provides a special confirmation procedure for print control. In order to use this procedure you must switch from automatic mode to confirmation mode. The following section describes the difference between automatic mode and confirmation mode.

Automatic mode - print output without print control

In automatic mode, openUTM controls the output from the printer. Output then proceeds as follows:

openUTM sends the first message in the queue to the printer and receives a positive or negative print confirmation from the printer.

If openUTM receives a positive print confirmation message, it deletes the message from the queue and sends the next message to the printer etc.

If openUTM receives a negative print confirmation message from the printer, it issues message K046. This message is not normally assigned to any specific UTM message destination. You can define a message destination for the message: the openUTM manual ”Messages, Debugging and Diagnostics” describes how to do this and indicates the destination to which you should assign the message.

The message destination for K046 can, for example, be the event service MSGTAC. Using the MSGTAC routine, which you have to create yourself (see the openUTM manual „Programming Applications with KDCS”), you can then respond to the error situation. The MSGTAC can, for example, switch on the confirmation mode. See also page 819.

Confirmation mode - output with print control

In confirmation mode, print output must be controlled by program units using PADM calls. Print outputs in confirmation mode proceed as follows:

openUTM sends a message to the printer. Once the message has been completed with a positive print confirmation, openUTM waits for confirmation, after which it performs a message termination procedure. The user/client can enter confirmation on the printer control LTERM or, for example, using the MSGTAC routine. For the MSGTAC routine, openUTM generates message K045 in response to a positive print confirmation message.

To confirm printer output, a procedure must be initiated using a PADM call which informs openUTM whether the print job should be repeated or whether it can now move on to print the next message.

With PADM AI, you can call up information about print jobs which you have to confirm. Users/clients on the printer control LTERM can therefore inform themselves about these messages and can also obtain information using the MSGTAC routine.

In confirmation mode, openUTM issues message K045 in response to a positive print confirmation message. You can assign message destination MSGTAC to this message: in this case, openUTM passes the message to the MSGTAC routine. The MSGTAC routine can then inform the printer control LTERM about the requested confirmation message.

Errors during print output (negative print confirmation messages) are handled in automatic mode.

Print control functions

openUTM provides print control functions with the PADM call. The actual function performed by PADM depends on the operation modifier which you send to openUTM in the *kcom* field of the parameter area. The following operation modifiers are available:

- PADM AC for switching on the confirmation mode
- PADM AT for switching off the confirmation mode. Automatic mode is reset.
- PADM PR for repeating a print output. The printer message is repeated on the same printer
- PADM OK for confirming print outputs
- PADM AI for calling up a list of print outputs to be confirmed with information

13.3.2.1 Activating/deactivating confirmation mode - PADM AC/AT

With PADM AC you can activate confirmation mode for one printer in the printer control LTERM or for all printers in a printer control LTERM. The print control function no longer runs automatically when in confirmation mode. openUTM does not delete the associated print job from the queue until a PADM OK call is stored for this printer.

PADM AT switches off the confirmation mode. Print output once again runs in automatic mode.

In UTM cluster applications, the change of confirmation mode applies globally to the cluster.

If PADM AT/AC is to operate on a specific printer, then you must specify the printer ID of that printer in the *kcrn* field. If the call is to apply for all printers in the printer control LTERM, you must enter blanks in *kcrn*.

Confirmation mode remains activated or deactivated beyond the termination of the current application.

When deactivating the confirmation mode, please note that any print output started while still in confirmation mode but not actually confirmed before the function was deactivated will still have to be confirmed in automatic mode. In other words, openUTM does not deal with subsequent print jobs for a given printer until a PADM OK has been issued.

KDCPADM MODE on page 835.

13.3.2.2 Confirming or repeating print output - PADM OK/PR

This function can only be used if confirmation mode is activated.
A print output job is confirmed with the call PADM OK. openUTM deletes the corresponding asynchronous job from the printer queue and can then deal with the next print job.

PADM PR repeats print output, for instance after a sample print run. The print job is not deleted from the queue. It remains at the front of the queue and is processed again.

See KDCPADM PRINT on page 836.

13.3.2.3 Querying information about print jobs to be confirmed - PADM AI

PADM AI provides information about print jobs to be confirmed. If there are no print jobs requiring confirmation, PADM AI simply returns blanks.

openUTM returns the following information about every print job:

– printer ID
– job ID of the asynchronous job
– user ID of the job submitter
– time the job was placed
– the target time for time-driven jobs
– positive and/or negative confirmation job

If you wish to query the print jobs requiring confirmation for all printers in the printer control LTERM, you must proceed as follows: when the first PADM/AI/PI call reaches the program unit, instead of a printer ID, send blanks in the *kcrn* field of the parameter area. openUTM then returns the printer ID of the first printer in the message area (together with other information). The printer ID of the next printer then appears in the *kcrmf* field. You then pass the contents of field *kcrmf* to field *kcrn* in the next PADM AI call etc. For the last printer, openUTM passes blanks in the *kcrmf* field

See KDCPADM INFORM, LIST =ACK on page 835.

13.3.3 Handling of errors during print output

Errors during print output are handled in the same way whether or not they have print control. This section describes which UTM features you can use to respond to printer malfunctions.

Hardware errors
The following action can be taken in response to hardware errors:

- The terminal assigned to the printer control LTERM is defective. When this happens, a different terminal can be assigned to the printer control LTERM by means of administration functions - such as with the administration command KDCSWTCH.
- A printer is defective. When this happens, a different printer can be assigned to the LTERM partner of the printer, and therefore to its message queue, for example using the KDCSWTCH command or by a procedure using the PADM CA call. If the LTERM partner is assigned to a printer control LTERM, then always ensure that the printer ID of the “new” printer is unique in the printer control LTERM area.

B *Formatting error*
B If errors occur when a logical message is being converted to a physical message (by
B VSTU), or to a formatted message (by FHS), UTM deletes the message and generates a
B dump. If the message is the main job in a complex of jobs, the negative confirmation job is
B started.

Error handling using MSGTAC routines

Targeted error handling is possible using the event service MSGTAC. Since the UTM program unit is authorized to perform administration work, it is capable of administering all printers in the application and of performing the print control function for all printers.

When errors occur, openUTM issues message K046. You can assign message destination MSGTAC to this message (see the openUTM manual ”Messages, Debugging and Diagnostics”). When this message appears, the MSGTAC routine is run. The MSGTAC routine can contain PADM calls. For example, it can:

– activate confirmation mode and then confirm or arrange for repetition of the print outputs from the printer control LTERM
– assign the LTERM partner of the printer, i.e. the queue for that printer, to a different printer
– inform a specific user/client about the error.

openUTM issues message K046 in response to the following errors:

– negative print confirmation message received from printer
– repetition of printer output
– not possible to establish a connection to the printer.

13.4 UTM program units for DADM and PADM functions

openUTM is supplied with the KDCS program units KDCDADM and KDCPADM. These provide you with all the services you will need for DADM and PADM calls without requiring you to generate your own program units for the administration of message queues and printers and for print control functions.

- KDCDADM provides the functions of DADM for the administration of messages.
- KDCPADM provides the functions of PADM for the administration of printers and for the control of message output to printers.

The ISP syntax tables required for KDCDADM and KDCPADM are present in KDCDAISP.

Procedures in which the program units KDCDADM and KDCPADM run function as interactive transactions in a dialog step. KDCDADM and KDCPADM expect to receive input in line mode: formatted input is rejected. The output generated by KDCDADM and KDCPADM are also issued in line mode.
KDCDADM and KDCPADM issue messages in English.

KDCDADM, KDCPADM and KDCDAISP are supplied as compiled objects or object modules. To enable you to use the program units together with the ISP syntax description, you must link them to your application program and record the program units and transaction codes used to boot the program units in the configuration of your application.

B B In openUTM on BS2000 systems the object modules are stored in the LMS library SYSLIB.UTM.064.EXAMPLE.

X X In openUTM on Unix and Linux systems you will find these objects in the library *libsample* under the path *utmpath/sample/sys*.

W W In openUTM on Windows systems you will find these objects in the library *utmpath*`\sys\libwork.lib`.

13.4.1 Generating KDCDADM and KDCPADM

The program units KDCDADM and KDCPADM must either be configured statically with KDCDEF or entered dynamically in the configuration. To enable you to use the functions of KDCDADM and KDCPADM you must assign dialog transaction codes to these program units. You can select any TAC name of your choice. In the following example, KDCDADM is assigned the transaction code *tacdadm* and KDCPADM is assigned the transaction code *tacpadm*.

Example of KDCDEF generation:

B – on BS2000 systems:

```
B       :
B       :
B   PROGRAM  KDCDADM,COMP=ILCS
B   PROGRAM  KDCPADM,COMP=ILCS
B   TAC      TACDADM,PROGRAM=KDCDADM,CALL=FIRST,TYPE=D
B   TAC      TACPADM,PROGRAM=KDCPADM,CALL=FIRST,TYPE=D
B       :
```

X/W – on Unix, Linux and Windows systems:

```
X/W     :
X/W     :
X/W PROGRAM  KDCDADM,COMP=C
X/W PROGRAM  KDCPADM,COMP=C
X/W TAC      tacdadm,PROGRAM=KDCDADM,CALL=FIRST,TYPE=D
X/W TAC      tacpadm,PROGRAM=KDCPADM,CALL=FIRST,TYPE=D
X/W     :
```

For KDCDEF generation in this application, you must also note the following:
The length of the standard primary working area specified in MAX SPAB= must be sufficient to accept the KDCS parameter area.

13.4.2 KDCDADM - Administer messages

The program unit KDCDADM makes it possible to administer messages in message queues. KDCDADM comprises three functions. You call up each of these functions by entering the transaction code which you assigned to program unit KDCDADM (called *tacdadm* from now on), together with a few operands. This next section describes which operands these should be.

KDCDADM covers the following functions:

- cancelling messages, i.e. deleting them from the message queue (DELETE)
- displaying information about messages in a message queue (INFORM)
- prioritizing a message, i.e. moving it to the front of the message queue (NEXT)
- moving messages from the dead letter queue (MOVE)

If you enter `tacdadm␣HELP`, openUTM informs you about the syntax of KDCDADM calls together with a brief description of the functions.

DELETE - Delete messages from the message queue

If you enter *tacdadm* together with the operand DELETE, you can delete messages from a message queue.

You can:

- Delete a specific message.
  To do this, you must provide unique identification for the message queue and the message. You identify the message queue, depending on the type, by means of the TAC name, the name of the LTERM partner, the user ID or the name of the temporary queue. You identify the message by means of its job ID and the time when the message was created. You can determine both of these items of data using the *tacdadm* INFORM.
- Delete all messages currently buffered in a message queue. This would delete all messages which are not yet being processed by the recipient (TAC, LTERM partner, user ID, temporary queue).

```
tacdadm␣   DELETE
           ,DESTINATION=destination
           [ ,DEST-TYPE = { LTERM | TAC | USER | QUEUE } ]
           ,DPUTID={ ALL | dput-id,GENTIME=time [,CHAINMSG= {ACT | DEL} ] }
```

DELETE Delete one message or all messages waiting in a message queue.

DESTINATION=destination
Specifies the message queue of the recipient containing the message to be canceled. For *destination* you must specify the name of a TAC, an LTERM partner, a user ID or the name of a temporary queue.

DEST-TYPE= Specifies the type of the recipient (*destination*). Possible entries are:

- LTERM The recipient is an LTERM partner.
- TAC The recipient is a TAC or a TAC queue.
- USER The recipient is the queue of a user ID.
- QUEUE The recipient is a temporary queue.

DPUTID= In DPUTID you specify which message is to be deleted.

ALL All messages to the recipient named in *destination* are to be deleted.

dputid One message in the queue is to be deleted. For *dputid* you must then specify the job ID for the message which is to be deleted.

GENTIME=time

You only have to enter this if you wish to delete one specific message from a queue (for DPUTID=*dputid*)
In this case, for *time* you must indicate the time at which the message was generated. Enter *time* in the form (ddd,hh,mm,ss) where *ddd* is the number of the day of the year, *hh* is the time in hours, *mm* the time in minutes and *ss* the time in seconds. openUTM requires *time* for unique identification of which message is to be deleted.

CHAINMSG= Indicates whether the negative confirmation job should be activated or not when deleting a job complex (DPUT job with confirmation jobs).

ACT The negative confirmation job is activated if it exists.

DEL The negative confirmation job is also deleted.

Default: ACT

Result

openUTM sends a message to the LTERM partner/LPAP partner through which the command was called. From the message you can identify whether the job was accepted or rejected. To find out whether openUTM was able to successfully execute the job, you must follow up with a KDCDADM INFORM query.

INFORM - Display information about message queues and messages

With *tacdadm* INFORM you can display information about message queues. UTM always provides the following items of information about individual messages in the queue:

– the job ID which you require, for example, when deleting a message
– the user ID with which the message was generated
– the time at which the message was generated
– with time-driven messages, the start time as of which the message should be processed
– information as to whether a positive or negative confirmation job belongs to the message.

In detailed information mode (LIST=LONG), openUTM also provides user information written with DPUT NI.

The lists containing the information returned by openUTM can be very extensive in some instances. For this reason, you have the options of:

– rerouting the output to a printer (OUT)
– restricting the output by specifying the job ID of the message at which the output list should start. The lists should be in ascending order of job ID. When you enter a job ID in CONT, the list starts with this message. No information is then provided about messages whose job ID occurs earlier in the alphabetic list.

```
tacdadm␣    INFORM
            ,DESTINATION=destination
            [ ,CONT=dputid ]
            [ ,DEST-TYPE = { LTERM | TAC | USER | QUEUE } ]
            [ ,LIST={ SHORT| LONG } ]
            [ ,OUT={ KDCDISP | ltermname } ]
```

INFORM Summary list of which messages in a message queue are to be output.

DESTINATION=destination
Name of the recipient of a message about which openUTM is to provide information. *destination* specifies the message queue. For *destination* you must specify the name of a TAC, an LTERM partner, a user ID or the name of a temporary queue.

DEST-TYPE= Specifies the type of the recipient (*destination*).Possible entries are:

LTERM or TAC
The recipient is a TAC, a TAC queue or an LTERM partner.

USER The recipient is the queue of a user ID.

QUEUE The recipient is a temporary queue.

CONT=dputid Controls the scope of output. For *dputid* you can enter the job ID of the message with which the list of information is to start. The list only contains information about messages whose job ID occurs later in the alphabet than *dputid* and about the message with the job ID specified in *dputid*.

LIST= Specifies the scope of information which openUTM is to output.

SHORT
The user information generated with DPUT NI is not output at the same time.

LONG The user information written with DPUT NI is not output at the same time.

Default: SHORT

OUT= Indicates where openUTM is to output the information.

KDCDISP openUTM outputs the information to the terminal at which the information was requested or openUTM passes the information to the client which requested the information.

ltermname openUTM outputs the information to a printer. For *ltermname*, enter the name of the LTERM partner assigned to the printer.

Result

For LIST=SHORT

```
User-id  DPUT-id    Gen-time   Start-time   Pos/Neg    Dest.
user1    dput-id    time1      time2        p/n/p n    dest1
```

Key to terms:

User-id
User ID or “*NONE“, if the user who generated the message has been deleted.

DPUT-id
Job ID of the message

Gen-time
: Time when the message was generated. Enter *time* in the following manner: (ddd,hh,mm,ss) where *ddd* is the number of the day in the year, *hh* is the time in hours, *mm* the time in minutes and *ss* the time in seconds.

Start-time
: This is output only for time-driven messages (DPUT messages).
Start-time is the earliest time as of which the job can be processed. The output format for time is the same as for *Gen-time*.

Pos/Neg
: Specifies whether a positive or negative confirmation job exists. The display field contains a “p” if a positive confirmation job exists and an “n” if a negative confirmation job exists. “p n” indicates that both a positive and a negative confirmation job exist.

Dest.
: Recipient of the message. For the dead letter queue, the original destination of the message is specified here, i.e. the name of an asynchronous TAC or a TAC queue. Otherwise, the field is empty.

For LIST=LONG

In addition to the information output for LIST=SHORT, the first 79 bytes of user information are output (in the next line - DPUT NI message). The following information appears on the output:

```
      User-id  DPUT-id  Gen-time   Start-time   Pos/Neg    Dest.
      user     dput-id  time1      time2        p/n/p n    dest1
User info:
   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

MOVE - Move messages from the dead letter queue

The dead letter queue is made up of messages which could not be processed.
In order to process these messages after any errors have been corrected, they must be assigned either to their original destination or to a new destination.

tacdadm MOVE allows you to move individual messages or all messages stored in the dead letter queue. The messages can be assigned to their original message queues, to any asynchronous TAC or to any TAC queue.

```
tacdadm␣   MOVE
            ,DESTINATION = { *ORIG | destination }
            ,DPUTID = { ALL | dputid, GENTIME = (ddd,hh,mm,ss) }
```

MOVE Move messages from the dead letter queue.

DESTINATION=
Specifies the new destination for the message.

*ORIG The message is to be assigned to its original destination.
If you specify DESTINATION=*ORIG together with DPUTID=ALL, all messages are assigned to their original destinations.

destination
Name of the new destination for the message or for all messages (asynchronous TAC or TAC queue).

DPUTID= ID of the message to be moved.

ALL All messages in the dead letter queue.

dputid Job ID of the message.

GENTIME=(ddd,hh,mm,ss)
Time the message was generated. Where:

ddd working day, *hh* hours, *mm* minutes, *ss* seconds.

Result

The job to move all messages to their original destinations is accepted without an error message being issued if individual original destinations or all the original destinations no longer exist.

openUTM generates a message indicating whether the job was accepted or not. The message is output at the terminal of the user issuing the job.

You must use separate KDCDADM demands in order to determine whether the messages have actually been moved.

The sample program DADMMVS or dadmmvsc for selectively moving messages from the dead letter queue is supplied with openUTM. The interactive program moves all messages from the dead letter queue using a specified original destination and a specified new destination. You can find the description of the sample program in the relevant system-specific openUTM manual “Using openUTM Applications”.

NEXT - Prioritize messages in the message queue

By entering *tacdadm* NEXT you can prioritize a message located anywhere in the message queue, moving it to the front of the queue. This makes the message you select the next message to be processed by the recipient.

You can only prioritize time-driven messages (DPUT jobs) if the specified start time (the earliest execution point) has already been reached.

```
tacdadm␣   NEXT
           ,DPUTID=dputid, GENTIME=(ddd,hh,mm,ss)
```

NEXT
: The message specified in *dputid* is placed in first position in the message queue.

DPUTID=dputid
: Job ID of the message to be prioritized.

GENTIME=(ddd,hh,mm,ss)
: Time when the message was generated;
ddd is the number of that day in the year, *hh* is the time in hours, *mm* the time in minutes and *ss* the time in seconds.

Result

openUTM generates a message which lets you know whether the job was accepted or not. The message is output to the terminal operated by the job submitter or is passed to the client which started the job.

13.4.3 KDCPADM - Print control and printer administration

The KDCPADM program unit enables you to administer printers and the control of print outputs. KDCPADM covers five functions. You can call each of these functions by entering the transaction code which you assigned to program unit KDCPADM (called *tacpadm* from now on) together with a few operands. This section describes which operands these are.

KDCPADM covers the following print control functions:

- Confirming print output or repeating an output item (PRINT)
- Switching between confirmation mode and automatic mode (MODE)

KDCPADM covers the following printer administration functions:

- Changing the status of a printer (STATE).
  You can disable a printer, re-enable a disabled printer, or establish/terminate a connection to a printer.

  In UTM cluster applications, the disabling and enabling of printers applies globally to the cluster.
- Assign a different or an additional printer to an LTERM partner, i.e. to a specific printer queue (SWITCH).
  This means that you can arrange for print jobs to be handled by another printer (e.g. in the event of a malfunction) or for a printer pool to be generated.

  This function is only permitted in standalone UTM applications.
- Inform about the printers assigned to a printer control LTERM (INFORM).

If you enter `tacpadm␣HELP`, openUTM informs you about the syntax of the KDCPADM call. openUTM provides a brief description of the functions.

INFORM - Display information about printers for a printer control LTERM

The *tacpadm* INFORM allows you to display information about printers and about the message queues assigned to any particular printer.

openUTM provides the following information about the printers assigned to the printer control LTERM:

– Name of the LTERM partner to which the printer is assigned
– Status of the printer, i.e. openUTM indicates whether the printer is currently connected to the application and whether or not the printer is disabled
– Confirmation mode, i.e. openUTM indicates whether the printer is set for automatic mode or confirmation mode
– Number of output jobs currently buffered in the queue of your selected printer, or the queue of the printer pool
– Number of time-driven output jobs currently buffered in the queue.

openUTM supplies the following information about the output jobs in the queue of a printer or a printer pool:

– Time at which the job was generated
– For time-driven jobs, the time as of which the job is to be processed
– Information about whether a positive or negative confirmation job is linked to the job.

The lists containing the information which openUTM returns can in some cases be very extensive. For this reason, the following options are provided:

– rerouting the output to a printer (OUT).
– restricting the output by specifying the job ID of the job with which the output list should start. The lists should be in ascending order of job ID. When you enter a job ID in CONT, the list starts with this job. No information is then provided about jobs whose job ID occurs earlier in the listing.

```
tacpadm␣    INFORM
            ,LIST= { PRINTERS | ACK }
            [ ,CID=cid1 ]
            [ ,CONT=cid2 ]
            [ ,OUT={ KDCDISP | ltermname1 } ]
            [ ,CTERM=ltermname ]
```

INFORM Outputs a summary list of printers or output jobs.

CID=cid1 (**c**ontrol-**ID**)
Printer ID of the printer. If you do not enter *cid* then openUTM returns information about all printers and message queues assigned to the printer control LTERM.

LIST= Indicates which information has been requested.

PRINTERS
Information about printers

ACK Information about the output jobs in the printer queues which still have to be confirmed.

CONT=cid2 Controls the scope of the output. For *cid2* you can enter the job ID of the printer as of which the list of information is to start.

There is no point entering CONT= unless the output produced in response to an earlier INFORM call does not fit on one screen page. To continue output, you enter the printer ID of the last printer that handled the previous output when you enter the next call in *cid2*.

OUT= Specifies where openUTM is to output the information.

KDCDISP openUTM outputs the information to the terminal on which the information was requested or passes the information to the client which requested the information.

ltermname1
openUTM outputs the information to the printer. For *ltermname1*, enter the name of the LTERM partner to which the printer is assigned.

Default: KDCDISP

CTERM=ltermname
Printer control LTERM to which the printer *cid1* belongs. For *ltermname*, enter the name of the printer control LTERM. If the command is not entered at the printer control LTERM of printer *cid1*, the user who enters the command must have administration privileges.

Default:
Name of the LTERM partner at which the command is entered.

Result

For LIST=PRINTERS

```
Control-id State   Connected Mode      LTERM-name  # of msg.: output delayed
cid1       Y/N     Y/N       auto/ack  lterm1                 num1   num2
```

Key to terms:

Control-id
: Printer ID of the printer

State
: Indicates whether the printer is disabled (N) or not (Y)

Connected
: Indicates whether the connection to the printer is established (Y) or not (N)

Mode
: Indicates whether confirmation mode (ack) or automatic mode (auto) is selected

LTERM name
: Name of the LTERM partner to which the printer is assigned

output
: Number of output jobs currently buffered in the printer queue

delayed
: Number of time-driven output jobs (DPUT job) currently buffered and waiting to be processed in the printer queue and whose start time has not yet been reached.

For for LIST=ACK:

```
Control-id   User-id  DPUT-id  Gen-time   Start-time   Pos/Neg chain msg
cid1         user1    dput-id  time1      time2         p / n / p n
```

Key to terms:

Control-id Printer ID of the printer

User-id
: User ID or “*NONE“, if the user who generated the job has been deleted.

DPUT-id
: Job ID of the asynchronous job

Gen-time
: Time at which the asynchronous job was generated. Enter *time* in the form (ddd,hh,mm,ss), where *ddd* is the number of the day in the year, *hh* is the time in hours, *mm* the time in minutes and *ss* the time in seconds.

Start-time
: This is output only for time-driven jobs (DPUT jobs).
Start-time is the earliest time as of which the job can be processed. The output format for time is the same as for *Gen-time*.

Pos/Neg chain msg.
: Specifies whether a positive or negative confirmation job exists. The display field contains a “p” if a positive confirmation job exists and an “n” if a negative confirmation job exists. “p n” indicates that both a positive and a negative confirmation job exist.

MODE - Change the confirmation mode for a printer

With *tacpadm* MODE you can change the confirmation mode. You can switch from automatic mode to confirmation mode and vice versa.

In UTM cluster applications, the change of confirmation mode applies globally to the cluster.

```
tacpadm␣   MODE
           [,CID=cid ]
             ,ACT={ ACK | AUTO}
           [ ,CTERM=ltermname ]
```

MODE Switches between automatic mode and confirmation mode for a printer.

CID=cid (**c**ontrol-**ID**)
Printer ID of the printer to be administered. If you do not enter *cid*, then the call addresses all printers assigned to the printer control LTERM *ltermname*. If the call is not placed at the printer control LTERM, then the user must have administration privileges.

ACT= Action to be performed, mandatory operand.

ACK Changes to confirmation mode, i.e. every print output has to be confirmed (e.g. with PRINT,...,ACT=NEXT).

AUTO Activates automatic mode, i.e. print output does not have to be confirmed. If *cid* is entered as a printer ID, the last print job for this printer is confirmed automatically.

CTERM=ltermname
Printer control LTERM, to which the printer *cid* belongs. For *ltermname*, please enter the name of the printer control LTERM. If the command is not entered at this printer control LTERM, then the user who started the procedure must have administration privileges.

Default:
Name of the LTERM partner at which the command is entered.

Result

openUTM returns a message informing you whether the job was accepted or rejected.

PRINT - Confirm / repeat print job

With *tacpadm* PRINT you can confirm a print job and arrange for the next job to be processed or for a print job to be repeated. In order to use the call *tacpadm* PRINT, confirmation mode must already be activated.

```
tacpadm␣   PRINT
           ,CID=cid
           [ ,ACT={ NEXT| REPEAT} ]
           [ ,CTERM=ltermname ]
```

PRINT Confirms or repeats print output

CID=cid (**c**ontrol-**ID**)
printer ID of the printer to which the call refers

ACT= Action to be performed:

- NEXT Print output is confirmed and the following output job is cleared for processing
- REPEAT Print output is to be repeated

  Default: NEXT

CTERM=ltermname
Name of the printer control LTERM to which the printer is assigned. If the command is not entered at this printer control LTERM, then the user who starts the procedure must have administration privileges.

Default:
Name of the LTERM partner at which the command was entered.

Result

openUTM returns a message informing you whether the job has been accepted or rejected.

STATE - Change the status of a printer

The *tacpadm* STATE allows you to change the status of a printer. You can:

- disable a printer or re-enable a disabled printer
- establish or terminate the connection to a printer.

```
tacpadm␣   STATE
           ,CID=cid
           ,ACT={ ON| OFF | CON | DIS | DISOFF }
           [ ,CTERM=ltermname ]
```

STATE — Changes the status of a printer

CID=cid — (**c**ontrol-**ID**)
Printer ID of the printer whose status is to be changed

ACT= — Action to be performed, mandatory operand.

- ON — Re-enable a disabled printer
- OFF — Disable a printer, i.e. it is no longer possible to establish a connection to this printer. If the printer is still connected at the time, the connection will be terminated.

  In UTM cluster applications, ON and OFF apply globally to the cluster.
- CON — Establish a connection to a printer
- DIS — Terminate the connection to a printer
- DISOFF — Terminate the connection to a printer and disable the printer.

CTERM=ltermname
Name of the printer control LTERM to which the printer is assigned. If the command is not entered at this printer control LTERM, the user who started the procedure must have administration privileges.

Default:
Name of the LTERM partner at which the command was entered.

Result

openUTM returns a message informing you whether the job has been accepted or rejected.

SWITCH - Change the assignment of printers to LTERM partners

The *tacpadm* SWITCH allows you to change the assignment of LTERM partners and printers. This function is only permitted in standalone UTM applications.

You can:

- assign the LTERM partner for this printer to a different printer, together with the message queue. This new printer then processes the print jobs in the queue sequentially. This enables you, for example, to print output jobs at a different printer if there is a malfunction on the original printer.
- to group printers together to form printer pools. This involves assigning several printers to one LTERM partner. All the printers in the pool will then work together to process the message queue for this LTERM partner. For more information about printer pools, see the openUTM manual “Generating Applications”.

```
tacpadm␣   SWITCH
           ,CID=cid
           ,LTERM=ltermname1
           [ ,CTERM=ltermname ]
```

SWITCH Changes the assignment of printers to LTERM partners

CID=cid (**c**ontrol-**ID**)
Printer ID of the printer to which a different LTERM partner is to be assigned.

LTERM=ltermname1
Name of the LTERM partner to which the printer is to be assigned. For *ltermname1* you can only enter an LTERM partner which has been specifically generated for printers and other output media. If a printer has already been assigned to the LTERM partner, this assignment is not terminated. The printers are simply grouped together to form a printer pool.

CTERM=ltermname
Name of the printer control LTERM to which the printer *cid* is assigned. If the command is not entered at this printer control LTERM, the user who starts the procedure must have administration privileges.

Default:
Name of the LTERM partner at which the command was entered.

Result

openUTM returns a message informing you whether the job has been accepted or rejected.

14 Appendix

14.1 Program interface for administration in COBOL

The COBOL program interface for administration purposes is very similar to the C/C++ program interface described in chapter 11. This means that you will also find it useful to refer to the description of the program interface in chapter 11 and to the descriptions dealing with the functional scope, the structure of user-defined administration programs, and central and automatic administration functions (chapters 3, 4, 5, 7, 8, 9 and 10) when writing your own administration programs in COBOL. This section lists the differences that you will need to be aware of when programming administration applications in COBOL:

The COBOL program interface differs from the C/C++ program interface in the following ways:

- In place of a header file (*kcadminc.h*) which includes all the data structures, COBOL is supplied with individual COPY members. Each of these COPY members usually contains only a single data structure (see table “COPY members for the program interface in COBOL” on page 840). This gives you the option of including individual data structures in programs which, under certain circumstances, can make programming considerably easier (e.g. when creating input/output tables).
- In accordance with COBOL conventions, field names use uppercase letters in place of lowercase letters and hyphens (-) in place of underscores (_).

  Example: The COBOL field name OBJ-TYPE corresponds to the C data field *obj_type*.

14.1.1 COPY members for the program interface in COBOL

The names of the COPY members for the program interface for administration are all prefixed with the letters KCA. The table below contains the names of the C data structures in alphabetical order and specifies which COPY member corresponds to which C data structure, or which COPY member contains particular definitions:

C data structure / definitions	COBOL COPY member
Operation codes and sub-operation codes for KDCADMI (values from *opcode* and *subopcode1/2)*, the object types (values from *obj_type*) and the main and subcodes of the return codes (values from *retcode*)	KCAOPRTC
Printable strings for the main and subcodes of the return codes	KCAPRINC
kc_abstract_syntax_str	KCAABSTC
kc_access_point_str	KCAACCPC
kc_adm_parameter (parameter area) and kc_id_area (identification area)	KCAPAIDC
kc_application_context_str	KCAAPLCC
kc_bcamappl_str	KCABCAMC
kc_change_application_str	KCAAPPLC
kc_cluster_curr_par_str	KCACCURC
kc_cluster_node_str	KCACLNOC
kc_cluster_par_str	KCACLPAC
kc_con_str	KCACONC
kc_create_statements_str	KCACREAC
kc_curr_par_str	KCACURRC
kc_db_info_str	KCADBIC
kc_diag_and_account_par_str	KCADACCC
kc_dyn_par_str	KCADYNC
B kc_edit_str	KCAEDITC
kc_encrypt_str	KCAENCRC
kc_encrypt_advanced_str	KCAENCAC
kc_gssb_str	[1]
kc_kset_str	KCAKSETC
kc_load_module_str	KCALMODC
kc_lock_mgtm_str	KCACLLKC
kc_lpap_str	KCALPAPC
kc_lses_str	KCALSESC

	C data structure / definitions	COBOL COPY member
	kc_lterm_str	KCALTRMC
	kc_ltac_str	KCALTACC
	kc_max_par_str	KCAMAXC
	kc_msg_dest_par_str	KCAMSGDC
	kc_message_module_str	KCAMSGMC
B	kc_mux_str	KCAMUXC
	kc_online_import_str	KCACLIMC
	kc_osi_association_str	KCAOASSC
	kc_osi_con_str	KCAOCONC
	kc_osi_lpap_str	KCAOLPAC
	kc_pagepool_str	KCAPGPLC
	kc_ptc_str	KCAPTCC
	kc_program_str	KCAPROGC
	kc_pterm_str	KCAPTRMC
	kc_queue_par_str	KCAQUPAC
	kc_queue_str	KCAQUEUC
	kc_sfunc_str	KCASFUNC
	kc_shutdown_str	KCASHUTC
	kc_signon_str	KCASIGNC
X/W	kc_subnet_str	KCASBNTC
	kc_syslog_str	KCASLOGC
	kc_system_par_str	KCASYSTC
	kc_tac_str	KCATACC
	kc_tacclass_str	KCATCLC
	kc_tasks_par_str	KCATASKC
	kc_timer_par_str	KCATIMEC
	kc_tpool_str	KCATPLC
	kc_transfer_syntax_str	KCATRANC
	kc_user_dyn1_str	KCAUSD1C
	kc_user_dyn2_str	KCAUSD2C
	kc_user_fix_str	KCAUSFXC

C data structure / definitions	COBOL COPY member
kc_user_str	KCAUSERC
kc_utmd_par_str	KCAUTMDC

[1] In this case there is no corresponding COPY member as kc_gssb_str only consists of the 8 character-long field in which the GSSB name (GS-NAME) is passed.

The COPY members for the COBOL program interface are stored in the following libraries:

B
- for openUTM on BS2000 systems: in the library SYSLIB.UTM.064.COB

X X X
- for openUTM on Unix and Linux systems: in the directory *utmpath*`/copy-cobol85` (Micro Focus Cobol compiler) or *utmpath*`/netcobol` (NETCOBOL compiler from Fujitsu)

W W W
- for openUTM on Windows systems: in the directory *utmpath*`\copy-cobol85` (Micro Focus Cobol compiler) or *utmpath*`\netcobol` (NETCOBOL compiler from Fujitsu)

14.1.2 KDCADMI function call

When calling KDCADMI you can - as with the C/C++ interface - pass four sets of parameters to openUTM: the parameter area KC-ADM-PARAMETER: the identification area ID-AREA; the selection area SELECT-AREA; and the data area DATA-AREA. To find out what data to supply to each of these areas, please refer to the description of the C/C++ interface in chapter 11. The KDCADMI must have the following syntax:

```
CALL "KDCADMI" USING KC-ADM-PARAMETER,
                     ID-AREA,
                     SELECT-AREA,
                     DATA-AREA.
```

14.1.3 Notes on programming

When writing administration programs in COBOL, please observe the following points:

- If you are working with the printable string tables for the return codes (the COPY member KCAPRINC), you will need to remember that, in COBOL, a table always begins with the index “1”, whereas return code values always begin with “0”. You can program accesses to a printable return code in the table as follows:

```
MOVE  MC-TEXT (KC-MAINCODE + 1) TO MC.
MOVE  SC-TEXT (KC-SUBCODE  + 1) TO SC.
```

- The data structure KC-ADM-PARAMETER with the call parameters for the administration interface begins on level 1. KC-ADM-PARAMETER therefore has to be stored in the WORKING-STORAGE-SECTION or the LOCAL-STORAGE-SECTION section.
- If a data structure contains substructures, then these should generally be addressed in fully qualified form.

  Example

```
MOVE SIGN-YEAR IN SIGN-TIME-DATE IN KC-USER-STR TO ...
```

14.2 Sample programs

Sample programs are shipped with the product openUTM in the form of source code and object modules. You can use these as programming templates for your own administration programs, modify them to suit your requirements, compile them and integrate them in your application. The sample programs are the programs HNDLUSR (only BS2000 systems), DADMMVS, PUBSUBA/PUBSUBD, SUSRMAX, COBUSER and ENCRADM (for a description of DADMMVS and PUBSUBA/PUBSUBD, see the relevant openUTM manual “Using openUTM Applications”).

B On BS2000 systems you will find the source code and the object modules of the sample programs, the ERRCHCK subroutine and the D0USER mask (IFG format) in the library SYSLIB.UTM.064.EXAMPLE.

X On Unix and Linux systems you will find the COBOL module `COBUSER.cbl` in the directory *utmpath*`/sample/src/mfcobol` or ...`/netcobol`. The C sample program and the subprogram ERRCHCK form part of the sample application. Following installation of the sample application, they can be found in the corresponding subdirectory `utmsample/utm-c`..

W On Windows systems you will find the COBOL module `COBUSER.cbl` in the directory *utmpath*`\sample\src\mfcobol` or ...`\netcobol`. The C sample program and the subprogram ERRCHCK for part of the Quick Start Kit. Following installation of the Quick Start Kit, they can be found in the corresponding subdirectory `\utmsample\utm-c`.

X/W The generation statements for the C sample programs are already entered in the KDCDEF input files of the sample application (Unix and Linux systems or of the Quick Start Kit (Windows systems).

B **The C program unit HNDLUSR**

B HNDLUSR allows you to carry out the following actions with format control:

B – query and modify the properties of user IDs
B – enter new user IDs in the configuration
B – delete user IDs from the configuration

B *Notes on generation*

B The program unit must be defined as follows in the KDCDEF run:

B `PROGRAM HNDLUSR,COMP=ILCS`

B `TAC HNDLUSR,PROGRAM=HNDLUSR,ADMIN=YES`

B `FORMSYS ENTRY=KDCFHS,TYPE=FHS,LIB=`*library with connection module to the formatting system*

B The program unit uses the C routine ERRCHCK and the FHS format D0USER internally.

B *Note on linking*

B The HNDLUSR program unit can be linked to the application program by means of a
B RESOLVE-BY-AUTO statement. The ERRCHCK routine is implicitly included as well.

B *Note on starting*

B The parameters for the FHS formatting system must be added to the start procedure for the
B application:

B `.FHS MAPLIB=`*format library*
B `.FHS ISTD=RUNP`

B You must copy the FHS format D0USER from the EXAMPLE library to the format library you
B are using before the application is started.

The C program unit SUSRMAX

You can use SUSRMAX to carry out the following actions:

- display all currently connected user IDs
- display all user IDs that are currently in a service
- display the currently set values for application parameters that can be defined in MAX at KDCDEF generation and modified by administration
- modify these application parameters

Notes on generation

The program unit must be defined as follows in the KDCDEF run:

B
```
PROGRAM SUSRMAX,COMP=ILCS
```
or

X/W
```
PROGRAM SUSRMAX,COMP=C

TAC SUSRMAX,PROGRAM=SUSRMAX,ADMIN=YES
```

The program unit requires the following minimum sizes for KB and SPAB:

– 168 bytes for the KB program area
– 6296 bytes for the SPAB area

The program unit uses the C routine ERRCHCK internally.

Note on linking

B B The SUSRMAX program unit can be linked to the application program by means of a RESOLVE-BY-AUTO statement. The ERRCHCK routine is also included implicitly.

X/W The SUSRMAX program unit is automatically linked into the example application.

The COBOL program unit COBUSER

The program reads information on signed-on users and LTERM partners.

Notes on generation

The program unit must be defined in the KDCDEF run as follows:

B `PROGRAM COBUSER, COMP=ILCS`
or
X/W `PROGRAM COBUSER, COMP=COB2` (Micro Focus compiler)
X/W `PROGRAM COBUSER, COMP=NETCOBOL` (NETCOBOL compiler from Fujitsu)

```
TAC COBUSER, PROGRAM=COBUSER, ADMIN=YES
```

Note on linking

B B The COBUSER program unit can be linked to the application program by means of a RESOLVE-BY-AUTO statement.

X X On Unix and Linux systems you must link the library `libsample` which is located below the path *utmpath*`/sample/sys`.

W On Windows systems, the COBUSER.obj object must be linked explicitly.

The C program unit ENCRADM

The ENCRADM program unit lets you perform the following administration functions for the encryption software.

– generate new RSA key pairs
– activate newly generated key pairs
– delete active and newly generated key pairs
– read out public keys in a file (both active and newly generated key pairs)

On Unix, Linux and Windows systems, ENCRADM is part of the sample application.

Notes on generation

The program unit must be defined using the following KDCDEF statements.

B
```
PROGRAM ENCRADM,COMP=ILCS
```
or

X/W
```
PROGRAM ENCRADM,COMP=C
```

```
TAC ENCRADM,PROGRAM=ENCRADM,ADMIN=YES
```

The program unit requires the following minimum space for KB and SPAB:
200 bytes for the KB program area and 4 KB for the SPAB area.

The program unit uses the C routine ERRCHCK internally.

Notes on linking

B
B
B The ENCRADM program unit is linked to the EXAMPLE library in the application program by means of a RESOLVE-BY-AUTO statement. The ERRCHCK routine is also linked implicitly.

X/W The ENCRADM program unit is automatically linked into the example application.

14.2.1 The C program units ADJTCLT

Using the C program unit ADJTCLT (adjust tac class), users can control how the processes (tasks) are distributed to the TAC classes in the light of the current total number of processes and the current number of asynchronous processes. To do this, the user creates a table containing the desired settings, see section "Creating a TAC class table".

The program is supplied as a full dialog and asynchronous program unit.

X/W
X/W On Unix, Linux and Windows systems, ADJTCLT forms part of the sample application or of the Quick Start Kit.

The program makes it possible to:

– Automatically adapt the number of TAC class processes in accordance with the table. This function is always executed.

– Read in a new table with a default name, see page 848. This function is executed if no table has as yet been read or if the operation code RF or READFILE is specified.

– Read in a new table with any name. This function is executed if the operation code RF or READFILE is specified.

– Modify the currently permitted number of asynchronous tasks. This function is executed if the operation code MA=*ttt* or MAXASYN=*ttt* is specified, where *ttt* is the desired maximum number of ASYNTASKS.

Because modifying the number of permitted tasks for asynchronous processing does not generate any messages, the change must not be made directly using the KDCAPPL command but must instead be performed via the interface provided by this program in order to adapt the TAC class settings.

If only one dialog TAC is generated for the program unit then all the functions must be started manually, i.e. the program must be called manually whenever the number of tasks or asynchronous tasks or the table has been changed.

For information on how to call the program automatically by means of the asynchronous TAC, see section “ADJTCLT as MSGTAC or MSG-DEST program unit” on page 851.

Creating a TAC class table

In this table, you specify the number of tasks per TAC class as a function of:

- the number of running processes
- and the current setting for the maximum number of processes that may be used for asynchronous processing.

The table must be saved as a text file. It can, for example, be created in Microsoft Excel and then be saved as a tab-separated text file. Spaces are also permitted as separators.

Sample table

A sample table with the following default name is supplied for all platforms:

B
- BS2000 systems: ADJTABLE.TXT

B
B
B
B
You will find this sample table in the library SYSLIB.UTM.064.EXAMPLE. Before the program unit can use this table, you must copy it to the user ID under which the UTM application is running. In UTM cluster applications, all the node applications can use the same table if the file is located in the shared pubset.

X/W
- Unix, Linux and Windows systems: AdjTable.txt

X/W
X/W
X/W
X/W
On Unix, Linux and Windows systems, the table forms part of the sample application or of the Quick Start Kit. Following the installation of the sample application or the Quick Start Kit, the table can be found in the following subdirectory of the sample application or Quick Start Kit:

X
W
`utmsample/utm-c` (Unix and Linux systems)
`utmsample\utm-c` (Windows systems)

Structure of the table

The following rules apply to the structure of the table:

- The values must be entered as printable numbers.
- The first row in the table can be a title line.
- All subsequent rows must have the following identical structure:
  - Column 1: Number of running processes. The maximum number of processes is 240.
  - Column 2: Current setting for the maximum number of processes that may be used for asynchronous processing.
  - Column 3 onwards: Number of processes for each TAC class in ascending order, e.g.

    ```
    17   2  4  3  3 ... 0  0  0  1  1  1
    ```

    Row 3 corresponds to TAC class 1, row 4 to TAC class 2 etc.

  The following applies to each of these rows:

  - The number of processes in row 1 must be greater than the total number of processes for all dialog TAC classes (1 - 8) plus the number of asynchronous processes. The greater this difference is, the more free processes there are for performing other tasks.
  - In the case of unused dialog TAC classes, it is also possible to specify 0 as the number of processes even though the minimum value for the number of processes for dialog TAC classes is 1. Reason: If this were not the case, the minimum number of processes for the application would be 9 (8 dialog TAC classes + 1).

    You should note that the program is not able to check whether these dialog TAC classes are genuinely unused.
  - The values for the TAC classes can also be omitted. In this case, the default value 0 is used for dialog TAC classes and the default value '-' for asynchronous TAC classes. '-' for asynchronous TAC classes means that the number of tasks for this TAC class is unchanged.
- The number of processes (column 1) must be sorted in ascending order in the table, while the numbers of asynchronous processes for which there are the same numbers of processes (column 2) must be sorted in descending order.
- Only the maximum permitted number of asynchronous processes, but not the number of processes permitted for the individual TAC classes, has an influence on the number of required processes. This is ignored if the number of permitted tasks for the individual asynchronous TAC classes is smaller than the maximum permitted number of asynchronous tasks.

Example

Extract from a table specifying that if there are 10 or fewer processes then one process, and if there are 12 or more processes then two processes must always be reserved, and in which only dialog TAC classes 1, 2 and 3 are used.

Alle	Asyn	Tcl01	Tcl02	Tcl03	…	Tcl11	Tcl12	Tcl13	Tcl14	Tcl15	Tcl16
4	0	1	1	1	…	0	0	0	0	2	2
5	1	1	1	1	…	0	0	0	0	1	2
5	0	2	1	1	…	0	0	0	0	1	1
6	2	1	1	1							
6	1	2	1	1	…	0	0	0	0	0	1
6	0	2	2	1	…	0	0	0	0	0	0
…											
10	5	2	1	1	…	0	0	0	0	3	3
10	3	3	2	1	…	0	0	0	0	1	2
10	1	3	3	2	…	–	–	–	–	–	–
10	0	4	3	2							
12	7	1	1	1	…	0	0	0	0	3	4
12	3	3	2	2	…	0	0	0	0	1	2
12	1	4	3	2							

The program starts the search at the last entry in the table and selects the table entry for which the following two conditions are satisfied:

– The number of running processes must be greater than or equal to the number in column 1.

– The difference between the number of currently running processes and the currently set maximum number of asynchronous tasks must be greater than or equal to column 1 - column 2.

  This condition ensures that the minimum number of processes that are free for dialog processing is actually greater than the sum of the numbers of processes in the dialog TAC classes.

If an entry is found then the program performs the following calculation:

```
Number of available dialog processes = Column 1(All) – Column 2 (Asyn)
```

Example

– If there are 12 processes and 7, 6, 5 or 4 asynchronous processes then the following row is selected:

```
12  7  1  1  1  …  0  0  0  0  3  4
```

– If there are 12 processes and 8 asynchronous processes then the following row is selected:

```
6   2  1  1  1
```

Of the 12 running processes, 4 are always free for dialog processing since a maximum of 8 processes are occupied by asynchronous processing. Of these 4 processes, a maximum of 3 are occupied by dialog processing (sum of the processes for the dialog TAC classes 1+1+1). There is therefore always one process free.

– If there are 12 processes and 9 asynchronous processes then no suitable entry is found.

If a suitable table entry is found then the number of processes for the individual TAC classes is adapted in accordance with the table.

In the case of TAC classes generated with PGWT=YES, the number of processes must not exceed the generated maximum number TASKS-IN-PGWT.

If the number of processes in TAC classes generated with PGWT=YES exceeds the maximum permitted number TASKS-IN-PGWT then the smaller value is used.

ADJTCLT as MSGTAC or MSG-DEST program unit

The asynchronous program variant of ADJTCLT can also be used as a MSGTAC or MSG-DEST program. An existing MSGTAC program can (after being adapted, if necessary) be integrated in the program unit. However, ADJTCLT cannot be used as a subprogram of an existing MSGTAC program.

ADJTCLT as MSGTAC program

If ADJTCLT is only generated as a MSGTAC program then only those functions that do not require an operation code are available since the MSGTAC program run is exclusively event-driven and the program cannot be called via a TAC.

In this case, there must be a separate application message module and the messages K052, K056 and K058 must have the message destination MSGTAC.

ADJTCLT as MSGDEST program

In the case of message-driven processing, the asynchronous TAC can also be generated as a MSG-DEST (MSG-DEST USER-DEST-1/2/3/4, NAME=, DEST-TYPE=TAC, …) and be used in the user message module as USER-DEST for messages K052, K056 and K058.

However, a message-driven program unit only runs independently of the number of asynchronous processes and TAC class control if it has been generated as a MSGTAC program. Otherwise the following applies:

– There must always be at least one process that is permitted to perform asynchronous processing.
– At least one process must be permitted in the program unit's TAC class.

Notes on generation

The program unit must be defined as follows in the KDCDEF run:

B
```
PROGRAM ADJTCLT, COMP=ILCS
```
X/W
```
PROGRAM ADJTCLT, COMP=C
```

Generation as dialog and asynchronous TAC:

```
TAC ADJTCLT, PROGRAM=ADJTCLT, ADMIN=YES, TYPE=D
TAC ADJTCLTA, PROGRAM=ADJTCLT, ADMIN=YES, TYPE=A
```

Generation as MSGTAC:

```
TAC KDCMSGTC, PROGRAM=ADJTCLT, ADMIN=YES, TYPE=A
```

The asynchronous TAC ADJTCLTA can be generated as follows as MSG-DEST for, e.g., USER-DEST-1.

```
MSG-DEST USER-DEST-1, NAME=ADJTCLTA, DEST-TYPE=TAC , MSG-FORMAT=PRINT
```

X/W X/W X/W On Unix, Linux and Windows systems, you can also take over the KDCDEF statements from the sample application or the Quick Start Kit into the generation of the UTM application.

Notes on linking

B B B On BS2000 systems, the program unit ADJTCLT can be linked to the application program by means of a RESOLVE-BY-AUTO statement. This also implicitly links the routine ERRCHCK.

X/W X/W On Unix, Linux and Windows systems, the program unit ADJTCLT is automatically linked to the sample application or the Quick Start Kit.

14.3 CALLUTM - Tool for administration and client/server communication

CALLUTM is a UPIC client on a BS2000 system which communicates with UTM applications that can be running either on the same BS2000 system or on a different system. CALLUTM can communicate with UTM applications irrespective of the operating system under which they happen to be running.

CALLUTM allows you, from within a BS2000 task, to start services in a UTM application, pass data to and receive data from those services. Messages are output in line mode. CALLUTM can run both in dialog mode and in batch mode, i.e. it can be implemented in procedural environments within a BS2000 task.

This makes CALLUTM particularly suitable for the central administration of local and remote UTM applications. With CALLUTM you can issue UTM administration commands and start administration programs in the UTM applications.

To do this, you must adapt the generations of the administered UTM applications, see section “Generation” on page 854.

To understand the following description of CALLUTM you will need to be familiar with UPIC on BS2000 systems, see the manual „openUTM-Client for the UPIC Carrier System”.

14.3.1 Instructions for use

14.3.1.1 Generation

B To use CALLUTM to administer UTM applications, proceed as follows:

B
B
B
- In the local BS2000 system: In the “side information file” (also known as the UPICFILE) for the UPIC carrier system, you must create the corresponding entries for the UTM applications (see the manual “openUTM-Client V4.0 for the UPIC Carrier System“).

B
B
B
- In each UTM application to be administered you must make PTERM entries and LTERM partner entries in the relevant configurations or generate an LTERM pool via which CALLUTM can connect.

B
B
B
B
B
- You need to create at least one user ID with administration privileges in each UTM application that you want to administer for this (see example below). CALLUTM must pass this user ID (along with the relevant password) to the UTM application when establishing the conversation. The CALLUTM statement CREATE-CONFIGURATION contains the operands USER-ID and PASSWORD for this (see page 866f).

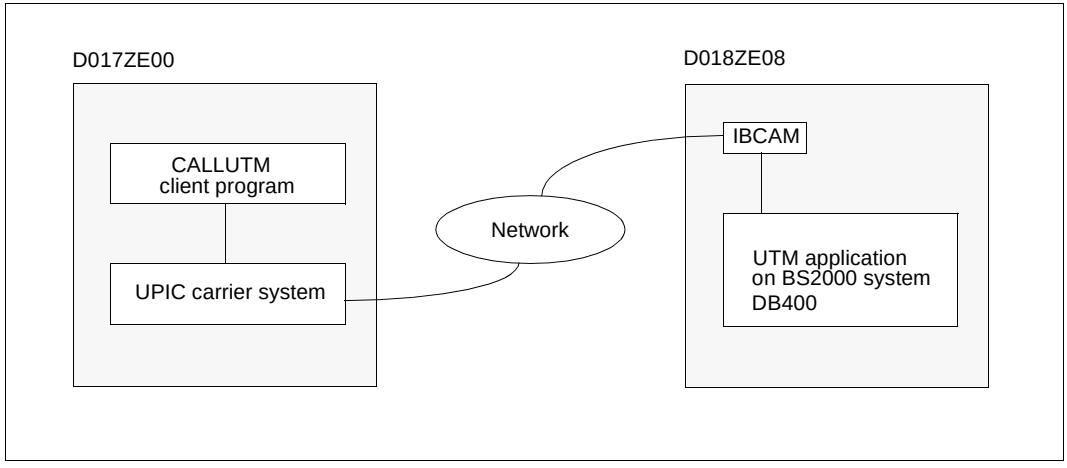
B
B
B
B
B
B

You can also assign the LTERM partner through which CALLUTM links up with the UTM application an user ID with administration privileges (LTERM ...,USER=). CALLUTM then does not need to pass a user ID to the UTM application and has administration privileges when establishing the connection. Bear in mind, however, that this approach will reduce access control for the UTM application.

B **Example**

B
B
The CALLUTM program on the BS2000 computer D017ZE00 is to communicate with the application DB400 on the BS2000 computer D018ZE08.

B



B Sample configuration for using the program CALLUTM

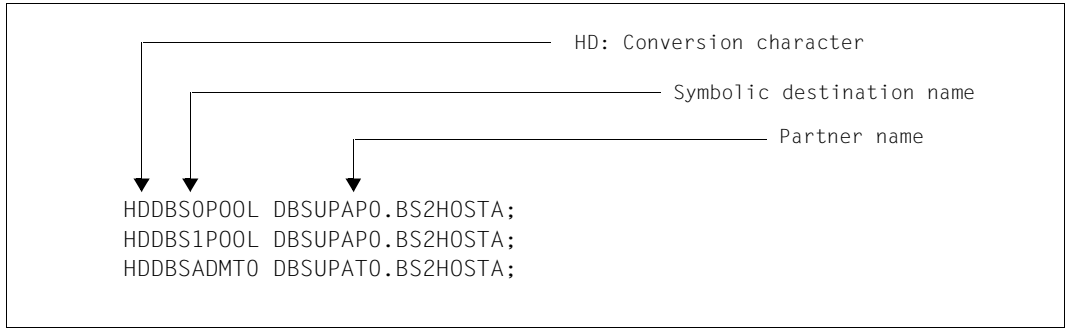
KDCDEF generation for the UTM application on the DB400 computer BS2HOSTA:

There are to be two ways in which CALLUTM can connect to the UTM application:

- via an LTERM pool. If CALLUTM connects via the LTERM pool, CALLUTM will be unable to start any administration commands and TACs for which administration privileges are required.
- via an LTERM partner generated explicitly for the purpose of working together with CALLUTM. To this end, a PTERM statement, an LTERM statement and a USER statement must be issued for CALLUTM in the UTM application.
- The user ID (USER ADMUPCT0) must have administrator privileges and be assigned to the LTERM partner.

```
*****
*- BCAMAPPL FOR CONNECTING CALLUTM VIA AN LTERM POOL
*****
BCAMAPPL DB4UPAP0,T-PROT=ISO
*****
*- BCAMAPPL FOR CONNECTING CALLUTM VIA A DEDICATED LTERM PARTNER
*****
BCAMAPPL DB4UPAT0,T-PROT=ISO
*****
*- LTERM POOL FOR CONNECTING CALLUTM ------------------------------*
*****
TPOOL BCAMAPPL=DB4UPAP0,KSET=ALLKEYS,LTERM=UPCP0#0,NUMBER=9,      -
      PRONAM=D017ZE00,PROTOCOL=N,PTYPE=UPIC-R
*****
*- DEFINE PTERM STATEMENT, LTERM PARTNER AND USER ID WITH --------*
*- ADMINISTRATION PRIVILEGES FOR CALLUTM --------------------------*
*****
PTERM UPCPT#T0, PRONAM=BS2HOSTC, PTYPE=UPIC-R,                    -
                BCAMAPPL=DB4UPAT0, PROTOCOL=N, LTERM=UPCLT#T0
LTERM UPCLT#T0, KSET=ALLKEYS, USER=ADMUPCT0, RESTART=N
* USER ID WITH ADMINSTRATOR PRIVILEGES ----------------------------
USER ADMUPCT0, PERMIT=ADMIN, PASS=(ADMT0   ,D)
*****
*-----------------------------------------------------------------*
*****
```

B *Entries in the UPICFILE*



B *Connecting CALLUTM to a UTM application*

- CALLUTM links up to the UTM application DB400 on the BS2HOSTA computer via the LTERM partner UPCLT#T0, if you make the following entries in CALLUTM when establishing the connection:

```
local name                = UPCPT#T0
symbolic destination name = DBSADMT0
```

„UPCPT#T0“ is passed to openUTM as the client name (PTERM name).

The symbolic partner name DBSADMT0 is linked to the partner name DB4UPAT0 from the UPICFILE. That is the name of the UTM application DB400, as specified in BCAMAPPL during the KDCDEF generation.

- CALLUTM links up to the UTM application via the LTERM pool, if you make the following entries when establishing the connection:

```
local name                = locname
symbolic destination name = DBS0POOL or DBS1POOL
```

Enter an alphanumeric name (up to 8 bytes long) for *locname* with which CALLUTM signs on with the transport system. This name is passed on to openUTM as the client name when the connection is established and it will be assigned to an LTERM partner of the LTERM pool for the duration of the connection.

The symbolic partner name DBS0POOL or DBS1POOL is linked to the partner name DB4UPAP0 from the UPICFILE. That is the name of the UTM application DB400 as entered in BCAMAPPL during the KDCDEF generation.

Calling CALLUTM as an administrator program in a BS2000 task

Before you use START-EXECUTABLE-PROGRAM to start the CALLUTM program, you must first use MODIFY-SDF-OPTION to assign the syntax file SYSSDF.UTM.064.CALLUTM (see also section “Execution” on page 876). Once the CALLUTM program run has terminated, you will need to deactivate the syntax file again.

Alternatively, you can call CALLUTM via the SDF command START-CALLUTM. This command can be found in the SDF UTM application area. For further information, see the openUTM manual “Using openUTM Applications on BS2000 Systems”, section “Calling UTM tools”.

Example (call via START-EXEC-PROG)

In the following example CALLUTM is to terminate the UTM application DB400 on the BS2HOSTA computer. For this you must first start CALLUTM and sign on to the UTM application DB400 using the user ID ADMUPCT0, which has administrator privileges. To terminate the application, you issue the UTM administration command KDCSHUT NORMAL.

To do that you must you must enter the following sequence of program statements. The program statements will be described in detail below.

```
/MODIFY-SDF-OPTION SYNTAX-FILE=*ADD(                                    -
/                              ADD-NAME=$user-id.SYSSDF.UTM.064.CALLUTM)
/ SET-FILE-LINK LINK-NAME = BLSLIB01                                    -
/              ,FILE-NAME = $TSOS.SYSLIB.SDF.047
/ SET-FILE-LINK LINK-NAME = BLSLIB02                                    -
/              ,FILE-NAME = $user-id.SYSLIB.UTM-CLIENT.064
/ SET-FILE-LINK LINK-NAME = BLSLIB03                                    -
/              ,FILE-NAME = $TSOS.SYSLNK.CRTE
/ START-PROGRAM *MOD($user-id.SYSLNK.UTM.064.CALLUTM,                    -
/                    CALLUTM,                                           -
/                    *ANY,                                              -
/                    *ADV(ALTERNATE-LIBRARIES = *YES) )
//CREATE-CONFIGURATION LOCAL-NAME=UPCPT#TO,                             -
/                               SYMB-DEST-NAME=DBSADMTO
//SELECT-SERVICE SERVICE-NAME=KDCSHUT, SERVICE-DATA='NORMAL'
//END
/MODIFY-SDF-OPTION SYNTAX-FILE=*REMOVE(                                 -
/                  REMOVE-NAME=$user-id.SYSSDF.UTM.064.CALLUTM)
```

By default, communication with the server is handled over a Socket connection. If CMX is to be used, you must issue the following sequence of statements before calling the program. The program statements used are described in detail in the next section.

```
/ SET-FILE-LINK LINK-NAME = BLSLIB01                                -
/              ,FILE-NAME = $TSOS.SYSLIB.SDF.047
/ SET-FILE-LINK LINK-NAME = BLSLIB02                                -
/              ,FILE-NAME = $user-id.SYSLIB.UTM-CLIENT.064.WCMX
/ SET-FILE-LINK LINK-NAME = BLSLIB03                                -
/              ,FILE-NAME = $user-id.SYSLIB.UTM-CLIENT.064
/ SET-FILE-LINK LINK-NAME = BLSLIB04                                -
/              ,FILE-NAME = $TSOS.SYSLNK.CRTE
/ SET-FILE-LINK LINK-NAME = BLSLIB05                                -
/              ,FILE-NAME = $TSOS.SYSLIB.CMX.014
```

An additional FILE-LINK is required before the program is called:

```
/ SET-FILE-LINK LINK-NAME = BLSLIB06                              -
/              ,FILE-NAME = $user-id.SYSLNK.UTM.064.SPLRTS
```

If communication to the server is to be encrypted, insert a SET-FILE-LINK command:

```
/ SET-FILE-LINK LINK-NAME = BLSLIB01                                -
/              ,FILE-NAME = $TSOS.SYSLIB.SDF.047
/ SET-FILE-LINK LINK-NAME = BLSLIB02                                -
/              ,FILE-NAME = $user-id.SYSLIB.UTM-CLIENT.064.WCMX
/ SET-FILE-LINK LINK-NAME=BLSLIB03,
                FILE-NAME=$user-id.SYSLNK.UTM-CL-CRYPT.064
/ SET-FILE-LINK LINK-NAME = BLSLIB04                                -
/              ,FILE-NAME = $user-id.SYSLIB.UTM-CLIENT.064
/ SET-FILE-LINK LINK-NAME = BLSLIB05                                -
/              ,FILE-NAME = $TSOS.SYSLNK.CRTE
/ SET-FILE-LINK LINK-NAME = BLSLIB06                                -
/              ,FILE-NAME = $TSOS.SYSLIB.CMX.014
```

14.3.2 Description of program statements

The program statements in CALLUTM are read by the SDF user interface and processed by the SDF command processor. Alongside the standard SDF statements, CALLUTM can also use any of the program statements listed in the table below:

No.	Program statement	Function
1.	CREATE-CONFIGURATION	Defines the environment for the program run and selects the connection to the UTM application. This statement must be issued as the first statement.
2.	SELECT-SERVICE	Starts a service (transaction code) in the UTM application. A message (operands or parameter values) can also be issued with this statement.
3.	CONTINUE-SERVICE	Continues a service that is not yet completed. For services which consist of a number of processing steps, this statement starts the next processing step once the previous one is completed. A message can also be issued with this statement.
4.	DEALLOCATE-CONVERSATION	Terminates the conversation with the UTM application. Any service that is still open in the UTM application and that belongs to this conversation will be terminated abnormally.
5.	SHOW-CONFIGURATION	Displays the program runtime environment that was set with CREATE-CONFIGURATION or MODIFY-CONFIGURATION.
6.	MODIFY-CONFIGURATION	Modifies the program runtime environment that was set with CREATE-CONFIGURATION.
7.	CALLUTM-ERROR-STEP	Controls statement processing of CALLUTM in procedure or batch mode.

CREATE-CONFIGURATION must always be the first statement to follow the program start. It is particularly important to ensure that CREATE-CONFIGURATION is issued before the SELECT-SERVICE statement.
Statements 5 and 6 can be issued anywhere between CREATE-CONFIGURATION and the end of the program run.

The statements are listed below in alphabetical order and described in detail.

Notational conventions

The following description of the statements uses SDF syntax notation. The table below outlines the elements that make up this form of notation, which are also described in the general section on “Notational conventions” on page 29.

Symbol	Meaning	Examples
< >	Angle brackets indicate variables whose values are described in terms of data types and the associated information.	**POSITION** = <integer 1..256>
/	The slash character separates operand values that can be used as alternatives.	**SET-TEST-MODE** = ***NO** / ***YES**
(…)	Parentheses indicate operand values which begin a structure.	,**SET-SERVICE-JV** = ***YES** (...) / ***NO**
Indentation	Indentation indicates dependency on the next highest operand.	,**SET-SERVICE-JV** = ***NO** / ***YES**(...) ***YES**(...) **JV-IDENTIFICATION** = **...**
|	A vertical line indicates operands that belong together in a structure. It runs from the start to the end of a structure. Further substructures can occur within a structure. The number of vertical lines to the left of an operand indicates the structural nesting depth.	,**SET-SERVICE-JV** = ***NO** / ***YES**(...) ***YES**(...) **JV-ID** = ***JV-NAME**(...) /... ***JV-NAME**(...) **JV-NAME** =...
Short name:	The following name is a guaranteed alias name for the statement name.	Short name: **CONFATTR**

CALLUTM-ERROR-STEP

The program statement CALLUTM-ERROR-STEP controls statement processing of CALLUTM in procedure or batch mode:

If an error (other than SDF syntax error) occurred during CALLUTM program run, e.g. if the addressed UTM application is offline, CALLUTM reads the following program statements from SYSDATA, until CALLUTM-ERROR-STEP is recognized. If no CALLUTM-ERROR-STEP is found, CALLUTM will terminate.

The CALLUTM-ERROR-STEP statement has no operands.

Example

```
//CREATE-CONFIGURATION ...
// ...
// ...
//SELECT-SERVICE SERVICE-NAME = KDCINF, SERVICE-DATA = C'STAT'       (1)
//SELECT-SERVICE SERVICE-NAME = KDXINF, SERVICE-DATA = C'USER'       (2)
//SELECT-SERVICE SERVICE-NAME = KDCINF, -                           (3)
// SERVICE-DATA = C'USER,L=KDCCON'
//CALLUTM-ERROR-STEP                                                (4)
//SELECT-SERVICE SERVICE-NAME = KDCINF, SERVICE-DATA = C'TAC'       (5)
```

Explanation:

Statement (1) is executed.
Statement (2) causes an error, because TAC KDXINF is not defined.
Statement (3) is not executed.
The processing is continued with statement (5).

B

CONTINUE-SERVICE

B CONTINUE-SERVICE allows you to continue a service that was started in the UTM application with SELECT-SERVICE and is made up of several steps. CONTINUE-SERVICE needs to be specified when the service sends a message to CALLUTM after one dialog step is completed but when the service as a whole is not yet completed because other processing steps remain to be executed. Data can be passed to the service for the next processing step.

B

CONTINUE-SERVICE

B **SERVICE-DATA** = **_*NO_** / list-poss(42):<c-string -with-lower-case 1..1800>

B ,**SET-SERVICE-JV** = **_*NO_** / ***YES**(...)

B ***YES**(...)
B **JV-IDENTIFICATION** = ***JV-NAME**(...) / ***LINK-NAME**(...)
B ***JV-NAME**(...)
B **JV-NAME**=<full-filename-without-generation-version 1..54>
B ,**POSITION** = **_1_** / <integer 1..256>
B ,**LENGTH** = **_*REST_** / <integer 1..256>
B *LINK-NAME
B **LINK-NAME** =< alphanum-name 1..7>
B ,**POSITION** = **_1_** / <integer 1..256>
B ,**LENGTH** = **_*REST_** / <integer 1..256>
B ,**PASSWORD** = **_*NONE_** /< c-string 1..4> / <x-string 1..8>
B ,**VALUE** = **_*RECEIVE-MSG_**(...) / <c-string-with-lower-case 1..256> / <x-string 1..512>
B ***RECEIVE-MSG**(...)
B **POSITION** = **_1_** / <integer 1..4000>

B For a description of the operands, see the SELECT-SERVICE statement (page 871f).

Example

This example refers to the sample administration program SUSRMAX which is supplied with openUTM (see “Sample programs” on page 41). The dialog with SUSRMAX consists of the following steps:

1. SUSRMAX is started and returns a message prompting you to select a function:

```
-> //SELECT-SERVICE SERVICE-NAME=SUSRMAX
<-
   date: 04-19-2016 time: 11:21:03

   application: DB400    host: BS2HOSTA tac: SUSRMAX
   ---------------------------------------------------------------------

   available commands:

   0 = end                          |  1 = show-connected-users
   2 = show-users-in-conversation   |  3 = show-changeable-max-values
   4 = change-max-values            |

   please make a selection
```

2. The service is continued with CONTINUE-SERVICE; the function “1 = show-connected-users” is selected (SERVICE-DATA=´1´). openUTM returns the requested information.

```
-> //CONTINUE-SERVICE SERVICE-DATA='1'
<-   ...
     ... Output
     ...
```

3. The UTM message prompting you to select another function is output again.

```
-> //CONTINUE-SERVICE
<-   ...
     ... The function selection message is output as in 1.
     ...
```

4. The service is continued with CONTINUE-SERVICE; the function ”2 = show-users-in-conversation” is selected (SERVICE-DATA=´2´). openUTM returns the requested information.

```
-> //CONTINUE-SERVICE SERVICE-DATA='2'
<-   ...
     ... Output
     ...
```

5. SUSRMAX is terminated (function “0 = end”).

```
B   -> //CONTINUE-SERVICE SERVICE-DATA='0'
B   <-
B      date: 04-19-2016 time: 11:22:34
B
B      application: DB400    host: D018ZE08 tac: SUSRMAX
B      ------------------------------------------------------------
B
B      conversation terminated
B
B      ------------------------------------------------------------
B
B   -> //
```

CREATE-CONFIGURATION

The statement CREATE-CONFIGURATION defines the environment for the program run and selects the connection to the UTM application. In other words, it allows you to determine:

- how the program is to sign on to the UPIC carrier system
- the UTM application to which a connection is to be established
- the UTM user ID to be passed when the conversation is established
- whether and to what extent a log file is to be written
- whether UPICTRACE is also to run.

CREATE-CONFIGURATION must be the first statement issued when the program has started. If CREATE-CONFIGURATION is issued repeatedly during the course of the program run, any open log files will be closed and open services will be rolled back. An internal DEALLOCATE is also executed.

You can use MODIFY-CONFIGURATION during a program run to modify the values set with CREATE-CONFIGURATION.

CREATE-CONFIGURATION Alias: **CONFATTR**

LOCAL-NAME = <alphanum-name 1..8>

,**SYMB-DEST-NAME** = <alphanum-name 8..8>

USER-ID = **<u>*NONE</u>** / <alphanum-name 1..8>(...) / <c-string_1..8_with-low> / < x-string 1..16>

<alphanum-name 1..8>(...) / <c-string_1..8_with-low> / < x-string 1..16>

PASSWORD = **<u>*NONE</u>** /< c-string 1..8 >/< x-string 1..16>

,**WRITE-LOGGING-FILE** = **<u>*NO</u>** / ***YES** (...)

***YES**(...)

LOGGING-FILENAME = <full-filename-without-generation-version 1..54>

,**OPEN-MODE** = ***REPLACE** / **<u>*EXTEND</u>**

,**LOGGING-INFO** = **<u>*ALL</u>** / ***SEND** / ***RECEIVE**

,**RECORD-LENGTH** =**<u>*STD</u>** / <integer 1..252>

,**WRITE-UPIC-TRACE** = **<u>*NO</u>** / ***YES**

,**CONFIGURATION-ID** = **<u>1</u>** / <integer 1..1>

,**SET-TEST-MODE** = **<u>*NO</u>** / ***YES**

,SET-ENCRYPTION-LEVEL = <u>*NO</u> / <integer 1..4>

LOCAL-NAME = <alphanum-name 1..8>
Local name under which CALLUTM signs on to the UPIC carrier system. This name must be defined in the UPICFILE.

SYMB-DEST-NAME = <alphanum-name 8..8>
Symbolic partner name of the UTM application to which a connection is to be established. This name must be defined in the UPICFILE.

USER-ID = UTM user ID used to establish the conversation.

*NONE No security functions are used.

<alphanum-name 1..8>( ) / <c-string_1..8_with-low> / < x-string 1..16>
UTM user ID. This user ID must exist in the UTM application.

PASSWORD = *NONE
No password is assigned to the user ID set for USER-ID.

PASSWORD = <c-string 1..8> or <x-string 1..16>
If a password is assigned to the user ID set for USER-ID, this password must be entered here as a character string (c-string) or as a hexadecimal string (x-string).

WRITE-LOGGING-FILE =
Determines whether and to what extent the flow of data from the client to the server and back (i.e. service data) is to be logged.

*NO No data is logged.

*YES ( ) Data is logged.

LOGGING-FILENAME = <full-filename-without-gen-vers 1..54>
Name of the log file.

OPEN-MODE = *EXTEND
Any existing log file is extended (appended). If no log file exists, a new one is created.

OPEN-MODE = *REPLACE
Any existing log file is overwritten. If no log file exists, a new one is created.

LOGGING-INFO = *SEND
Only data that is sent to a service in the UTM application is logged.

LOGGING-INFO = *RECEIVE
Only the data that CALLUTM receives from a service in the UTM application is logged.

LOGGING-INFO = *ALL
All data that CALLUTM exchanges with a service in the UTM application is logged, i.e. both data that is sent and data that is received.

RECORD-LENGTH = *STD / <integer 1..252>
Length of the records to be written to the log file. A new record is started whenever “newline” is detected in the send or receive area. “newline” itself is not logged.
*STD stands for a record length of 80 bytes.

WRITE-UPIC-TRACE =
Indicates whether UPIC tracing is to be activated.

*NO The UPIC trace is not activated.

*YES The UPIC trace is activated.
If no *UPICTRA link name to a job variable exists, a new job variable JV.UPICTRACE.CALLUTM is created with the value “-SX” and the link name *UPICTRA is assigned to it.

CONFIGURATION-ID = 1
Serves to identify the configuration. The only permissible value is 1.

SET-TEST-MODE=
Activate/deactivate test mode.

*NO Test mode is not activated. No UPIC calls are output to SYSOUT.

*YES Test mode is activated. All UPIC calls from CALLUTM are output to SYSOUT.

SET-ENCRYPTION-LEVEL=
Specifies whether and how data is to be encrypted over the connection.

If encryption has been generated on the server side for the client connection, the relevant encryption level from the generation must be specified here.

If encryption has been generated on the server side for a TAC that is to be called here, the encryption level must also be specified on the client side.

*NO No encryption.

<integer 1..4>
Encryption level from the generation.

Example

```
//CREATE-CONFIGURATION LOCAL-NAME=UPCPT#TO,SYMB-DEST-NAME=DBSADMTO, -
//                     WRITE-LOGGING-FILE=*YES(L-F=LOG.CALLUTM)
```

DEALLOCATE-CONVERSATION

This statement terminates the conversation with the partner application. Any service that is still open in the UTM application is terminated abnormally.

DEALLOCATE-CONVERSATION
CONFIGURATION-ID = **1** / <integer 1..1>

CONFIGURATION-ID does not need to be specified (default setting).
Only CONFIGURATION-ID=1 may be specified.

Example

```
//DEALLOCATE-CONVERSATION
```

MODIFY-CONFIGURATION

MODIFY-CONFIGURATION allows you to modify the existing values set with CREATE-CONFIGURATION or a previous MODIFY-CONFIGURATION statement in the program runtime environment.

MODIFY-CONFIGURATION Alias: **MODATTR**

LOCAL-NAME = *UNCHANGED / <alphanum-name 1..8>
,**SYMB-DEST-NAME** = ***UNCHANGED** / <alphanum-name 8..8>
,**USER-ID** = ***UNCHANGED** / <alphanum-name 1..8>
<alphanum-name 1..8>(...)
PASSWORD = ***UNCHANGED** / ***NONE** /< c-string 1..8> / <x-string 1..16>
,**WRITE-LOGGING-FILE** = ***UNCHANGED** / ***YES** (...)
***YES**(...)
LOGGING-FILENAME = ***UNCHANGED** / <full-filename-without-generation-version 1..54>
,**OPEN-MODE** = ***UNCHANGED** / ***REPLACE** / ***EXTEND**
,**LOGGING-INFO** = ***UNCHANGED** / ***ALL** / ***SEND** / ***RECEIVE**
,**RECORD-LENGTH** = ***UNCHANGED** / ***STD** /< integer 1..252>
,**WRITE-UPIC-TRACE** = ***UNCHANGED** / ***NO** / ***YES**
,**CONFIGURATION-ID** = **1** / <integer 1..1>
,**SET-TEST-MODE** = ***UNCHANGED** / ***NO** / ***YES**
,SET-ENCRYPTION-LEVEL = ***UNCHANGED** / ***NO** / <integer 1..4>

For a description of the operands, see statement CREATE-CONFIGURATION on page 866.

Example

Logging is deactivated.

```
-> //MOD-CONF WRITE-LOGGING-FILE=*NO
<- CUA0050: configuration modified
-> //
```

SELECT-SERVICE

SELECT-SERVICE starts a service in the UTM application. Data (i.e. operands and parameter values) required by the service for processing can also be supplied. In addition, a job variable can be defined to accept the receive message, part of the receive message or any specified string once the statement has been executed. If the job variable defined has not yet been cataloged a new one is created.

If you use SELECT-SERVICE to call a service which consists of several processing steps and which passes dialog messages to CALLUTM between the individual processing steps, then, between the time the service is called and the time when it is terminated, you can (with the exception of the standard SDF statements) only issue the following statements:

- CONTINUE-SERVICE
  Continues the service once a dialog message has been received.
- DEALLOCATE-CONVERSATION
  Aborts the connection and (abnormally) terminates the service in the UTM application.
- SHOW-CONFIGURATION
  Reads the current configuration data.

SELECT-SERVICE

SERVICE-NAME = <alphanum-name 1..8>/<c-string-with-lower-case>
,**SERVICE-DATA** = **_*NO_** / list-poss(42): <c-string -with-lower-case 1..1800>
,**SET-SERVICE-JV** = **_*NO_** / ***YES**(...)
***YES**(...)
JV-IDENTIFICATION = ***JV-NAME**(...) / ***LINK-NAME**(...)
***JV-NAME**(...)
JV-NAME = <full-filename-without-generation-version 1..54>
,**POSITION** = **_1_** /< integer 1..256>
,**LENGTH** = **_*REST_** /< integer1..256>
***LINK-NAME**(...)
LINK-NAME = <alphanum-name 1..7>
,**POSITION** = **_1_** /< integer 1..256>
,**LENGTH** = **_*REST_** **/** < integer 1..256>
,**PASSWORD** = **_*NONE_** /< c-string 1..4> / <x-string 1..8>
,**VALUE** = **_*RECEIVE-MSG_**(...) /< c-string-with-lower-case 1..256> /< x-string 1..512>
***RECEIVE-MSG**(...)
POSITION = **_1_** /< integer 1..4000>

SERVICE-NAME = <alphanum-name 1..8>/<c-string-with-lower-case>
The transaction code with which the service is to be started in the UTM application; can also be an administration command or a different administration TAC.

SERVICE-DATA =
Message to be passed to the service in the UTM application.

*NO No message data is passed.

list-poss(42): <c-string-with-lower-case 1..1800>
Message to be passed from the remote service. The message must be passed as a C string, i.e. enclosed in quotes.
You can also pass a list of C strings here. The individual elements of the list are sent as partial messages and also received as partial messages by the server.
Number of C strings: up to 42.
Total length of the list: up to 1800 characters

SET-SERVICE-JV =
Supplies data to a job variable.

*NO Data is not supplied to a job variable.

*YES ( ) Data is supplied to a job variable.

JV-IDENTIFICATION = *JV-NAME ( )
The job variable is addressed by name.

JV-NAME = <full-filename-without-generation-version 1..54>
Name of the job variable. If the job variable has not yet been cataloged a new one is created.

POSITION = 1
The job variable is set as of column 1.

POSITION = <integer 1..256>
The job variable is set as of the specified column.

LENGTH = *REST
As of POSITION, the full-length job variable can be set.

LENGTH = <integer 1..256>
The job variable is set to the specified length
(depending on the input value and starting position).

JV-IDENTIFICATION = *LINK-NAME ( )
The job variable is addressed via a link name which must have been set before the statement was executed (e.g. at the start of the program run).

LINK-NAME = <alphanum-name 1..7>
Link name set for the job variable.

POSITION = as for *JV-NAME ( ); see above

LENGTH = as for *JV-NAME ( ); see above

PASSWORD = *NONE
Access to the job variable is not password-protected.

PASSWORD = <c-string 1..4> / <x-string 1..8>
Password used for (read and write) access to the job variable.

VALUE = *RECEIVE-MSG (POSITION = 1/<integer 1..4000>)
The job variable is reserved with the data received from the UTM application (in accordance with the position and length defined above).
For POSITION you specify the position (column) within the receiving area as of which the data received is to be written to the job variable. If POSITION ≠1, the job variable is positioned at the corresponding distance within the receiving area.

VALUE = <c-string 1..256> / <x-string 1..512>
The job variable is reserved with the string passed for this value in accordance with the position and length defined above.

Note on the use of job variables

Before the statement is executed, the job variable is initialized with blanks as of the specified position and to the specified length. If an error occurs during this access to the job variable, the statement as a whole is not executed.

Example

1. The administration command KDCSHUT WARN, TIME=01 is called.

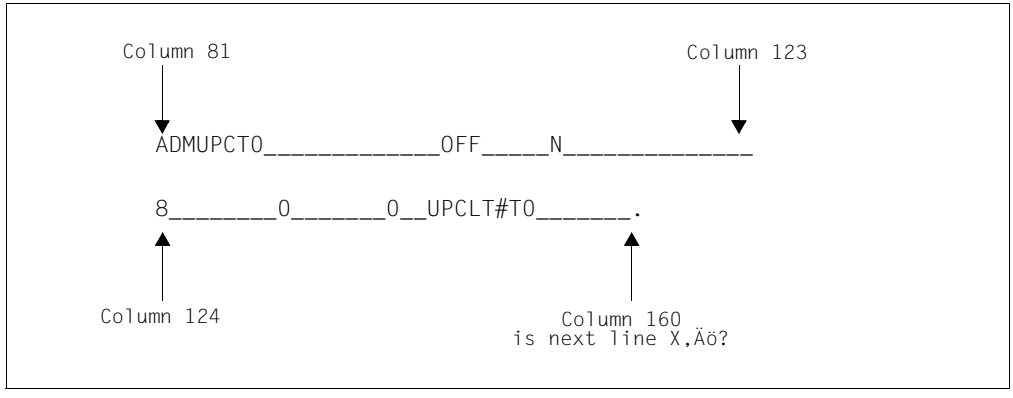
```
//SELECT-SERVICE SERVICE-NAME=KDCSHUT, SERVICE-DATA='WARN,TIME=01'
```

2. The command KDCINF reads the properties of the user ID UPCUSER (KDCINF USER,LIST=UPCUSER). The output is to be written to the job variable JV.USER (as of column 81).

```
//SELECT-SERVICE SERVICE-NAME=KDCINF,                        -
//               SERVICE-DATA='USER,LIST=(UPCUSER)',       -
//               SET-SERVICE-JV=*YES (                       -
//                      JV-ID=*JV-NAME  (                    -
//                         JV-NAME=JV.USER,                  -
//                         POSITION=81  )                    -
//                                     )                     -
//                      VALUE=*RECEIVE-MSG(POSITION=161)
```

Result

Once the statement has executed, the job variable JV.USER will be reserved as follows as of column 81:



For an explanation of the meaning of the contents, see also page 740:

UPCUSER (value of USER):
Name of the user ID.

OFF (value of STATUS):
The user ID is disabled.

N (value of OSERV):
The user ID is not processing any service at present.

8 (value of NR.TACS):
Eight transaction jobs have so far been entered under this user ID.

0 (value of SECCNT):
Number of security violations under this user ID.

UPCLT#T0 (value of LTERM):
Name of the LTERM partner via which the user ID signs on.

SHOW-CONFIGURATION

SHOW-CONFIGURATION allows you to display the values defined by the last CREATE-CONFIGURATION or MODIFY-CONFIGURATION statement.

SHOW-CONFIGURATION	Alias: SHOWATTR
CONFIGURATION-ID = **1** / <integer_1..1> ,**OUTPUT** = ***SYSOUT**/***LOGGING-FILE**	

CONFIGURATION-ID = 1
: Identifies the configuration. The only permissible value is 1.

OUTPUT= Specifies the destination to which the requested data is to be output.

*SYSOUT
: The requested data is to be output to SYSOUT.

*LOGGING-FILE
: The requested data is to be written to the log file (see CREATE-CONFIGURATION; operand WRITE-LOGGING-FILE). If no log file was defined with CREATE-CONFIGURATION or MODIFY-CONFIGURATION, the data output will be rerouted to SYSOUT.

Example

```
-> //SHOW-CONF OUTPUT=*SYSOUT
<- -- current configuration data: ----------------------------------
   local name = UPCPT#TO
   symbolic destination name = DBSADMTO
   partner name (from upicfile) = DBSUPATO
   program name is enabled to UPIC
   no user identification given
   logging file name = LOG.CALLUTM
      open mode = replace
      record length =  80
      logging info = transmitted and received messages
      file is open
   upic trace is switched off
   program monitoring job variable is not specified
encryption is not available
-- end configuration data ----------------------------------------
->
```

14.3.3 Components, system environment, software configuration

The following components are supplied for CALLUTM:

- the program SYSPRG.UTM.064.CALLUTM
- the SDF syntax file SYSSDF.UTM.064.CALLUTM

The program CALLUTM is contained in the LMS library SYSLNK.UTM.064.CALLUTM. It requires the following software configuration:

- BS2000 systems with OSD/BC as of V8.0
- CMX(BS2000) as of V1.4 if CMX is to be used to communication
- SDF as of V4.7C
- JV as of V15.0A (job variables)

The job variables are used with the link names UPICFIL, UPICPAT and UPICTRA as described in the openUTM manual „openUTM-Client for the UPIC Carrier System”.

14.3.4 Integration in a UTM application

To enable the program CALLUTM to communicate with a UTM application, entries and definitions along the lines of those shown in the example in section “Instructions for use” on page 854 should be applied - with suitable modifications - to the current application.

14.3.5 Execution

The following section is applicable if you have used START-EXEC-PROG to call CALLUTM. If you perform the call using the SDF command START-CALLUTM then the correct syntax file is assigned automatically.

If the syntax file SYSSDF.UTM.064.CALLUTM has not yet been assigned when the program starts, the program will automatically look for it within the user ID under which it is currently running or under the user ID specified in the job variable linked by means of *UPICPAT. If the program is unable to assign the syntax file, the program is aborted with an error message to this effect.

The syntax file can be assigned with the following command:

```
/MODIFY-SDF-OPTION SYNTAX-FILE=*ADD(                          -
/                    ADD-NAME=[$userid.]SYSSDF.UTM.064.CALLUTM)
```

The name of the syntax file, i.e. SYSSDF.UTM.064.CALLUTM, must not be changed.

14.3.6 Program-monitoring job variables

If the program is started with a program-monitoring job variable (i.e. with the MONJV operand in the call), then CALLUTM supplies the following values to the job variable in addition to the values set by the operating system:

Column	Length	Contents	Meaning
129	1	D / P / B	Mode in which CALLUTM is running. D: CALLUTM is running in dialog mode P: CALLUTM is running in a procedure B: CALLUTM is running in a batch job
131 - 134	4	<tsn>	Task sequence number of the job
136 - 139	4	<nnnn>	Serial number of the statement within the current program run (standard SDF statements are not counted); leading zeroes are suppressed.
141 - 148	8	CUA<name>	Internal name of the most recently executed statement or the current statement. The following values can occur for <name>: CREA for CREATE-CONFIGURATION MODC for MODIFY-CONFIGURATION SHOWC for SHOW-CONFIGURATION SELS for SELECT-SERVICE CONTS for CONTINUE-SERVICE DEALL for DEALLOCATE-CONVERSATION
150	1	C / N / O	Status of service processing: C: A service is still open in the server application and must be continued with CONTINUE-SERVICE. N: No more services are open. O: A service was called but no message has yet been received from it.
152 - 159	8	<servname>	Name of the service in the UTM application (transaction code)
161 - 168	8	<localnam>	“Local name”: the name under which CALLUTM is currently signed on to the UPIC carrier system.
170 - 177	8	<symbdest>	“Symbolic destination name” of the UTM application to which CALLUTM is currently connected or is currently establishing a connection (name as defined in the UPICFILE).
179 - 210	32	<partner>	Name of the UTM application linked in the UPICFILE to the “symbolic destination name”.
251 - 256	6	<nnnnnn>	Number of the error that caused the program to terminate; leading zeroes are suppressed. 0 indicates normal termination.

14.3.7 Messages issued by CALLUTM

CALLUTM generates the following messages:

```
CUA0010: give attributes
```

This message is output after CALLUTM has been started and prompts you to enter the statement CREATE-CONFIGURATION.

```
CUA0015: symb-dest-name not found in UPICFILE
```

This message can occur after either of the statements CREATE-CONFIGURATION or MODIFY-CONFIGURATION has been issued. It is output if the name specified in the operand SYMB-DEST-NAME does not exist in the UPICFILE. The statement is aborted.
In procedures, this also causes the program to abort.

```
CUA0020: conversation started by service <name> has to be continued
```

This message indicates that a service started previously with SELECT-SERVICE needs to be continued. You can now issue the CONTINUE-SERVICE statement to continue the service, or abort it with DEALLOCATE-CONVERSATION.
<name> outputs the transaction code with which the service was started.

```
CUA0025: error analysing SDF statement = nnnn
```

An error occurred during analysis of an SDF statement. nnnn indicates the place within the program at which the error occurred.
In procedures, this causes the program to abort.

```
CUA0030: JV = <name> not accessible (error = nnnnnnn)
```

The error nnnnnnn occurred during initialization of the job variable <name>. The program attempts to rectify the error. If it cannot do so this will, in procedures, cause the program to abort.

```
callutm:error in <upic-call>:n
CUA0035: error in send-receive routine = nnnn
```

An error occurred during a call to the UPIC carrier system. Before the message CUA0035 is output, CALLUTM first issues „`callutm:...`“ regarding the UPIC call (<upic-call>) during which the error occurred and the UPIC return code that was generated n (for an explanation of the meaning, see the manual „openUTM-Client for the UPIC Carrier System”).

```
callutm:error in <upic-call>:n
```

In CUA0035, nnnn indicates the place within the program at which the error occurred.
In procedures, this error causes the program to abort.

Example

```
//**- callutm: error in allocate: 1 -**//
CUA0035: error in send-receive routine = 2007
```

This means that it was not possible to establish a connection to the UTM application. This message is output if the UTM application is not available, if the BCMAP commands issued for the UTM application contained errors, or if no BCMAP commands have yet been issued for the UTM application.

```
CUA0040: not processed statement = <name>
CUA0045: program run is continued with next statement
```

The statement <name> could not be executed. The program run is not aborted: you can continue it by issuing further statements. In procedures, the program run is continued with the following statement:

```
CUA0050: configuration modified
```

This message can occur after a MODIFY-CONFIGURATION statement. It indicates that the statement was executed successfully and that the configuration has been modified as specified. You can now resume your work under the new configuration.

```
CUA0051: no value given to modify configuration
```

This message can occur after a MODIFY-CONFIGURATION statement. It indicates that the statement was analyzed successfully, but that the values specified did not result in a modification of the configuration.

```
CUA0055: conversation deallocated and abnormal end
```

The conversation with the UTM application has been shut down. The service in the UTM application has been aborted.
In procedures, this causes the program to abort.

Possible causes:

– The service in the UTM application encountered an error (PEND ER).

– An administration service was started for which the client did not possess the necessary privileges.

```
CUA0060: no logging file assigned, output re-assigned to sysout (stdout)
```

This message can occur after a SHOW-CONFIGURATION statement for which OUTPUT=*LOGGING-FILE was specified to request output to a log file.
The message indicates that no log file has yet been assigned; instead, output is redirected to SYSOUT.

```
CUA0065: deallocate not executed (program not in conversation),
         program run can be continued with next statement"
```

The DEALLOCATE-CONVERSATION statement has been entered but no service was open.

The output is sent to SYSOUT.

```
CUA0070: restart not possible, continue with new service
```

It was not possible to restart with KDCDISP.

```
CUA0080: for a list of c-strings the total c-string-length (1800) is exceeded
```

A list of C strings was entered in the SERVICE-DATA operand in the SELECT-SERVICE or CONTINUE-SERVICE statement. The length of the data exceeds the maximum permitted length.

```
CUA0085: current conversation will be terminated
```

CALLUTM is running in a procedure or in batch mode and the UPIC transport protocol has reported an error. CALLUTM terminates the open service and may then branch to the statement CALLUTM-ERROR-STEP or reaches the end of the statements.

```
CUA0090: encryption is not available in this environment
```

Encryption not available.

Action: Integrate encryption in the current UPIC client library.

```
CUA0100: CALLUTM-ERROR-STEP reached
```

After the occurrence of an error in a procedure or in batch mode, all statements were skipped until CALLUTM-ERROR-STEP was recognized.

```
CUA0105: all statements will be ignored until CALLUTM-ERROR-STEP is
recognized
```

After the occurrence of an error in a procedure or in batch mode, all statements are skipped until CALLUTM-ERROR-STEP is detected. If no such statement is found, END terminates the program run.

Glossary

A term in *italic* font means that it is explained somewhere else in the glossary.

abnormal termination of a UTM application
: Termination of a *UTM application*, where the *KDCFILE* is not updated. Abnormal termination is caused by a serious error, such as a crashed computer or an error in the system software. If you then restart the application, openUTM carries out a *warm start*.

abstract syntax (OSI)
: Abstract syntax is defined as the set of formally described data types which can be exchanged between applications via *OSI TP*. Abstract syntax is independent of the hardware and programming language used.

acceptor (CPI-C)
: The communication partners in a *conversation* are referred to as the *initiator* and the acceptor. The acceptor accepts the conversation initiated by the initiator with Accept_Conversation.

access list
: An access list defines the authorization for access to a particular *service*, *TAC queue* or *USER queue*. An access list is defined as a *key set* and contains one or more *key codes*, each of which represent a role in the application. Users or LTERMs or (OSI) LPAPs can only access the service or *TAC queue/USER queue* when the corresponding roles have been assigned to them (i.e. when their *key set* and the access list contain at least one common *key code)*.

access point (OSI)
: See *service access point.*

ACID properties
: Acronym for the fundamental properties of *transactions*: atomicity, consistency, isolation and durability.

administration
: Administration and control of a *UTM application* by an *administrator* or an *administration program*.

administration command

Commands used by the *administrator* of a *UTM application* to carry out administration functions for this application. The administration commands are implemented in the form of *transaction codes*.

administration journal

See *cluster administration journal*.

administration program

Program unit containing calls to the *program interface for administration*. This can be either the standard administration program *KDCADM* that is supplied with openUTM or a program written by the user.

administrator

User who possesses administration authorization.

AES

AES (Advanced Encryption Standard) is the current symmetric encryption standard defined by the National Institute of Standards and Technology (NIST) and based on the Rijndael algorithm developed at the University of Leuven (Belgium). If the AES method is used, the UPIC client generates an AES key for each session.

Apache Axis

Apache Axis (Apache eXtensible Interaction System) is a SOAP engine for the design of Web services and client applications. There are implementations in C++ and Java.

Apache Tomcat

Apache Tomcat provides an environment for the execution of Java code on Web servers. It was developed as part of the Apache Software Foundation's Jakarta project. It consists of a servlet container written in Java which can use the JSP Jasper compiler to convert JavaServer pages into servlets and run them. It also provides a fully featured HTTP server.

application cold start

See *cold start*.

application context (OSI)

The application context is the set of rules designed to govern communication between two applications. This includes, for instance, abstract syntaxes and any assigned transfer syntaxes.

application entity (OSI)

An application entity (AE) represents all the aspects of a real application which are relevant to communications. An application entity is identified by a globally unique name (“globally” is used here in its literal sense, i.e. worldwide), the *application entity title* (AET). Every application entity represents precisely one *application process*. One application process can encompass several application entities.

application entity qualifier (OSI)

Component of the *application entity title*. The application entity qualifier identifies a *service access point* within an application. The structure of an application entity qualifier can vary. openUTM supports the type “number”.

application entity title (OSI)

An application entity title is a globally unique name for an *application entity* (“globally” is used here in its literal sense, i.e. worldwide). It is made up of the *application process title* of the relevant *application process* and the *application entity qualifier*.

application information

This is the entire set of data used by the *UTM application*. The information comprises memory areas and messages of the UTM application including the data currently shown on the screen. If operation of the UTM application is coordinated with a database system, the data stored in the database also forms part of the application information.

application process (OSI)

The application process represents an application in the *OSI reference model*. It is uniquely identified globally by the *application process title*.

application process title (OSI)

According to the OSI standard, the application process title (APT) is used for the unique identification of applications on a global (i.e. worldwide) basis. The structure of an application process title can vary. openUTM supports the type *Object Identifier*.

application program

An application program is the core component of a *UTM application*. It comprises the main routine *KDCROOT* and any *program units* and processes all jobs sent to a *UTM application*.

application restart

see *warm start*

application service element (OSI)

An application service element (ASE) represents a functional group of the application layer (layer 7) of the *OSI reference model*.

application warm start

see *warm start*.

association (OSI)

An association is a communication relationship between two application entities. The term “association” corresponds to the term *session* in *LU6.1*.

asynchronous conversation

CPI-C conversation where only the *initiator* is permitted to send. An asynchronous transaction code for the *acceptor* must have been generated in the *UTM application*.

asynchronous job

Job carried out by the job submitter at a later time. openUTM includes *message queuing* functions for processing asynchronous jobs (see *UTM-controlled queue* and *service-controlled queue)*. An asynchronous job is described by the *asynchronous message*, the recipient and, where applicable, the required execution time. If the recipient is a terminal, a printer or a transport system application, the asynchronous job is a *queued output job*. If the recipient is an *asynchronous service* of the same application or a remote application, the job is a *background job*. Asynchronous jobs can be *time-driven jobs* or can be integrated in a *job complex*.

asynchronous message

Asynchronous messages are messages directed to a *message queue*. They are stored temporarily by the local *UTM application* and then further processed regardless of the job submitter. Distinctions are drawn between the following types of asynchronous messages, depending on the recipient:

– In the case of asynchronous messages to a *UTM-controlled queue,* all further processing is controlled by openUTM. This type includes messages that start a local or remote *asynchronous service* (see also *background job*) and messages sent for output on a terminal, a printer or a transport system application (see also *queued output job*).
– In the case of asynchronous messages to a *service-controlled queue*, further processing is controlled by a *service* of the application. This type includes messages to a *TAC queue*, messages to a *USER queue* and messages to a *temporary queue*. The USER queue and the temporary queue must belong to the local application, whereas the TAC queue can be in both the local application and the remote application.

asynchronous program

Program unit started by a *background job*.

asynchronous service (KDCS)

Service which processes a *background job*. Processing is carried out independently of the job submitter. An asynchronous service can comprise one or more program units/transactions. It is started via an asynchronous *transaction code*.

audit (BS2000 systems)

During execution of a *UTM application*, UTM events which are of relevance in terms of security can be logged by *SAT* for auditing purposes.

authentication

See *system access control*.

authorization

See *data access control*.

Axis

See *Apache Axis*.

background job

Background jobs are *asynchronous jobs* destined for an *asynchronous service* of the current application or of a remote application. Background jobs are particularly suitable for time-intensive processing or processing which is not time-critical and where the results do not directly influence the current dialog.

basic format

Format in which terminal users can make all entries required to start a service.

basic job

Asynchronous job in a *job complex*.

browsing asynchronous messages

A *service* sequentially reads the *asynchronous messages* in a *service-controlled queue*. The messages are not locked while they are being read and they remain in the queue after they have been read. This means that they can be read simultaneously by different services.

bypass mode (BS2000 systems)

Operating mode of a printer connected locally to a terminal. In bypass mode, any *asynchronous message* sent to the printer is sent to the terminal and then redirected to the printer by the terminal without being displayed on screen.

cache

Used for buffering application data for all the processes of a *UTM application*. The cache is used to optimize access to the *page pool* and, in the case of UTM cluster applications, the *cluster page pool*.

CCS name (BS2000 systems)

See *coded character set name*.

client

Clients of a *UTM application* can be:

– terminals
– UPIC client programs
– transport system applications (e.g. DCAM, PDN, CMX, socket applications or UTM applications which have been generated as *transport system applications*).

Clients are connected to the UTM application via LTERM partners.
openUTM clients which use the OpenCPIC carrier system are treated just like *OSI TP partners*.

client side of a conversation

This term has been superseded by *initiator*.

cluster

A number of computers connected over a fast network and which in many cases can be seen as a single computer externally. The objective of clustering is generally to increase the computing capacity or availability in comparison with a single computer.

cluster administration journal

The cluster administration journal consists of:

– two log files with the extensions JRN1 and JRN2 for global administration actions,
– the JKAA file which contains a copy of the KDCS Application Area (KAA). Administrative changes that are no longer present in the two log files are taken over from this copy.

The administration journal files serve to pass on to the other node applications those administrative actions that are to apply throughout the cluster to all node applications in a UTM cluster application.

cluster configuration file

File containing the central configuration data of a *UTM cluster application*. The cluster configuration file is created using the UTM generation tool *KDCDEF*.

cluster filebase

Filename prefix or directory name for the *UTM cluster files*.

cluster GSSB file

File used to administer GSSBs in a *UTM cluster application*. The cluster GSSB file is created using the UTM generation tool *KDCDEF*.

cluster lock file

File in a *UTM cluster application* used to manage cross-node locks of user data areas.

cluster page pool

The cluster page pool consists of an administration file and up to 10 files containing a *UTM cluster application's* user data that is available globally in the cluster (service data including LSSB, GSSB and ULS). The cluster page pool is created using the UTM generation tool *KDCDEF*.

cluster start serialization file

Lock file used to serialize the start-up of individual node applications (only on Unix systems and Windows systems).

cluster ULS file

File used to administer the ULS areas of a *UTM cluster application*. The cluster ULS file is created using the UTM generation tool *KDCDEF*.

cluster user file

File containing the user management data of a *UTM cluster application*. The cluster user file is created using the UTM generation tool *KDCDEF*.

coded character set name (BS2000 systems)

If the product *XHCS* (e**X**tended **H**ost **C**ode **S**upport) is used, each character set used is uniquely identified by a coded character set name (abbreviation: “CCS name” or “CCSN”).

cold start

Start of a *UTM application* after the application terminates normally (*normal termination*) or after a new generation (see also *warm start*).

communication area (KDCS)

KDCS *primary storage area*, secured by transaction logging and which contains service-specific data. The communication area comprises 3 parts:

– the KB header with general service data
– the KB return area for returning values to KDCS calls

– the KB program area for exchanging data between UTM program units within a single *service*.

communication resource manager

In distributed systems, communication resource managers (CRMs) control communication between the application programs. openUTM provides CRMs for the international OSI TP standard, for the LU6.1 industry standard and for the proprietary openUTM protocol UPIC.

configuration

Sum of all the properties of a *UTM application*. The configuration describes:

– application parameters and operating parameters
– the objects of an application and the properties of these objects. Objects can be *program units* and *transaction codes*, communication partners, printers, *user IDs*, etc.
– defined measures for controlling data and system access.

The configuration of a UTM application is defined at generation time (*static configuration*) and can be changed dynamically by the administrator (while the application is running, *dynamic configuration*). The configuration is stored in the *KDCFILE*.

confirmation job

Component of a *job complex* where the confirmation job is assigned to the *basic job*. There are positive and negative confirmation jobs. If the *basic job* returns a positive result, the positive confirmation job is activated, otherwise, the negative confirmation job is activated.

connection bundle

see *LTERM bundle*.

connection user ID

User ID under which a *TS application* or a *UPIC client* is signed on at the *UTM application* directly after the connection has been established. The following applies, depending on the client (= LTERM partner) generation:

– The connection user ID is the same as the USER in the LTERM statement (explicit connection user ID). An explicit connection user ID must be generated with a USER statement and cannot be used as a “genuine” *user ID*.

– The connection user ID is the same as the LTERM partner (implicit connection user ID) if no USER was specified in the LTERM statement or if an LTERM pool has been generated.

In a *UTM cluster application*, the service belonging to a connection user ID (RESTART=YES in LTERM or USER) is bound to the connection and is therefore local to the node.
A connection user ID generated with RESTART=YES can have a separate service in each *node application*.

contention loser

Every connection between two partners is managed by one of the partners. The partner that manages the connection is known as the *contention winner*. The other partner is the contention loser.

contention winner

A connection's contention winner is responsible for managing the connection. Jobs can be started by the contention winner or by the *contention loser*. If a conflict occurs, i.e. if both partners in the communication want to start a job at the same time, then the job stemming from the contention winner uses the connection.

conversation

In CPI-C, communication between two CPI-C application programs is referred to as a conversation. The communication partners in a conversation are referred to as the *initiator* and the *acceptor*.

conversation ID

CPI-C assigns a local conversation ID to each *conversation*, i.e. the *initiator* and *acceptor* each have their own conversation ID. The conversation ID uniquely assigns each CPI-C call in a program to a conversation.

CPI-C

CPI-C (Common Programming Interface for Communication) is a program interface for program-to-program communication in open networks standardized by X/Open and CIW (**C**PI-C **I**mplementor's **W**orkshop).
The CPI-C implemented in openUTM complies with X/Open's CPI-C V2.0 CAE Specification. The interface is available in COBOL and C. In openUTM, CPI-C can communicate via the OSI TP, *LU6.1* and UPIC protocols and with openUTM-LU62.

Cross Coupled System / XCS

Cluster of BS2000 computers with the *Highly Integrated System Complex* Multiple System Control Facility (HIPLEX® MSCF).

data access control

In data access control openUTM checks whether the communication partner is authorized to access a particular object belonging to the application. The access rights are defined as part of the configuration.

data space (BS2000 systems)

Virtual address space of BS2000 which can be employed in its entirety by the user. Only data and programs stored as data can be addressed in a data space; no program code can be executed.

dead letter queue

The dead letter queue is a TAC queue which has the fixed name KDCDLETQ. It is always available to save queued messages sent to transaction codes or TAC queues but which could not be processed. The saving of queued messages in the dead letter queue can be activated or deactivated for each message destination individually using the TAC statement's DEAD-LETTER-Q parameter.

DES

DES (Data Encryption Standard) is an international standard for encrypting data. One key is used in this method for encoding and decoding. If the DES method is used, the UPIC client generates a DES key for each session.

dialog conversation

CPI-C conversation in which both the *initiator* and the *acceptor* are permitted to send. A dialog transaction code for the *acceptor* must have been generated in the *UTM application*.

dialog job, interactive job

Job which starts a *dialog service*. The job can be issued by a *client* or, when two servers communicate with each other (*server-server communication*), by a different application.

dialog message

A message which requires a response or which is itself a response to a request. The request and the response both take place within a single service. The request and reply together form a dialog step.

dialog program

Program unit which partially or completely processes a *dialog step*.

dialog service

Service which processes a *job* interactively (synchronously) in conjunction with the job submitter (*client* or another server application) . A dialog service processes *dialog messages* received from the job submitter and generates dialog messages to be sent to the job submitter. A dialog service comprises at least one *transaction.* In general, a dialog service encompasses at least one dialog step. Exception: in the event of *service chaining*, it is possible for more than one service to comprise a dialog step.

dialog step

A dialog step starts when a *dialog message* is received by the *UTM application*. It ends when the UTM application responds.

dialog terminal process (Unix , Linux and Windows systems)

A dialog terminal process connects a terminal of a Unix, Linux or Windows system with the work processes of the *UTM application*. Dialog terminal processes are started either when the user enters utmdtp or via the LOGIN shell. A separate dialog terminal process is required for each terminal to be connected to a UTM application.

Distributed Lock Manager / DLM (BS2000 systems)

Concurrent, cross-computer file accesses can be synchronized using the Distributed Lock Manager.
DLM is a basic function of HIPLEX® MSCF.

distributed processing

Processing of *dialog jobs* by several different applications or the transfer of *background jobs* to another application. The higher-level protocols *LU6.1* and *OSI TP* are used for distributed processing. openUTM-LU62 also permits distributed processing with LU6.2 partners. A distinction is made between distributed processing with *distributed transactions* (transaction logging across different applications) and distributed processing without distributed transactions (local transaction logging only). Distributed processing is also known as server-server communication.

distributed transaction

Transaction which encompasses more than one application and is executed in several different (sub)-transactions in distributed systems.

distributed transaction processing

Distributed processing with *distributed transactions.*

dynamic configuration

Changes to the *configuration* made by the administrator. UTM objects such as *program units*, *transaction codes*, *clients*, *LU6.1 connections*, printers or *user IDs* can be added, modified or in some cases deleted from the configuration while the application is running. To do this, it is necessary to create separate *administration programs* which use the functions of the *program interface for administration*. The WinAdmin administration program or the WebAdmin administration program can be used to do this, or separate *administration programs* must be created that utilize the functions of the *administration program interface*.

encryption level

The encryption level specifies if and to what extent a client message and password are to be encrypted.

event-driven service

This term has been superseded by *event service*.

event exit

Routine in an application program which is started automatically whenever certain events occur (e.g. when a process is started, when a service is terminated). Unlike *event services*, an event exit must not contain any KDCS, CPI-C or XATMI calls.

event function

Collective term for *event exits* and *event services*.

event service

Service started when certain events occur, e.g. when certain UTM messages are issued. The *program units* for event-driven services must contain KDCS calls.

filebase

UTM application filebase
On BS2000 systems, filebase is the prefix for the *KDCFILE*, the *user log file* USLOG and the *system log file* SYSLOG.
On Unix, Linux and Windows systems, filebase is the name of the directory under which the KDCFILE, the user log file USLOG, the system log file SYSLOG and other files relating to to the UTM application are stored.

generation

See *UTM generation*.

global secondary storage area

See *secondary storage area*.

hardcopy mode

Operating mode of a printer connected locally to a terminal. Any message which is displayed on screen will also be sent to the printer.

heterogeneous link

In the case of *server-server communication:* a link between a *UTM application* and a non-UTM application, e.g. a CICS or TUXEDO application.

Highly Integrated System Complex / HIPLEX®

Product family for implementing an operating, load sharing and availability cluster made up of a number of BS2000 servers.

HIPLEX® MSCF

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)
Provides the infrastructure and basic functions for distributed applications with HIPLEX®.

homogeneous link

In the case of *server-server communication*: a link between two *UTM applications.* It is of no significance whether the applications are running on the same operating system platforms or on different platforms.

inbound conversation (CPI-C)

See *incoming conversation.*

incoming conversation (CPI-C)

A conversation in which the local CPI-C program is the *acceptor* is referred to as an incoming conversation. In the X/Open specification, the term “inbound conversation” is used synonymously with “incoming conversation”.

initial KDCFILE

In a *UTM cluster application*, this is the *KDCFILE* generated by *KDCDEF* and which must be copied for each node application before the node applications are started.

initiator (CPI-C)

The communication partners in a *conversation* are referred to as the initiator and the *acceptor*. The initiator sets up the conversation with the CPI-C calls Initialize_Conversation and Allocate.

insert

Field in a message text in which openUTM enters current values.

inverse KDCDEF

A function which uses the dynamically adapted configuration data in the *KDCFILE* to generate control statements for a *KDCDEF* run. An inverse KDCDEF can be started “offline” under *KDCDEF* or “online” via the *program interface for administration*.

JDK

Java Development Kit
Standard development environment from Oracle Corporation for the development of Java applications.

job

Request for a *service* provided by a *UTM application*. The request is issued by specifying a transaction code. See also: *queued output job*, *dialog job*, *background job*, *job complex*.

job complex

Job complexes are used to assign *confirmation jobs* to *asynchronous jobs*. An asynchronous job within a job complex is referred to as a *basic job*.

job-receiving service (KDCS)

A job-receiving service is a *service* started by a *job-submitting service* of another server application.

job-submitting service (KDCS)

A job-submitting service is a *service* which requests another service from a different server application (*job-receiving service*) in order to process a job.

KDCADM

Standard administration program supplied with openUTM. KDCADM provides administration functions which are called with transaction codes (*administration commands*).

KDCDEF

UTM tool for the *generation* of *UTM applications*. KDCDEF uses the configuration information in the KDCDEF control statements to create the UTM objects *KDCFILE* and the ROOT table sources for the main routine *KDCROOT*.
In UTM cluster applications, KDCDEF also creates the *cluster configuration file*, the *cluster user file*, the *cluster page pool*, the *cluster GSSB file* and the *cluster ULS file*.

KDCFILE

One or more files containing data required for a *UTM application* to run. The KDCFILE is created with the UTM generation tool *KDCDEF*. Among other things, it contains the *configuration* of the application.

KDCROOT

Main routine of an *application program* which forms the link between the *program units* and the UTM system code. KDCROOT is linked with the *program units* to form the *application program*.

KDCS message area

For KDCS calls: buffer area in which messages or data for openUTM or for the *program unit* are made available.

KDCS parameter area

See *parameter area*.

KDCS program interface

Universal UTM program interface compliant with the national DIN 66 265 standard and which includes some extensions. KDCS (compatible data communications interface) allows dialog services to be created, for instance, and permits the use of *message queuing* functions. In addition, KDCS provides calls for *distributed processing*.

Kerberos

Kerberos is a standardized network authentication protocol (RFC1510) based on encryption procedures in which no passwords are sent to the network in clear text.

Kerberos principal

Owner of a key.
Kerberos uses symmetrical encryption, i.e. all the keys are present at two locations, namely with the key owner (principal) and the KDC (Key Distribution Center).

key code

Code that represents specific access authorization or a specific role. Several key codes are grouped into a *key set*.

key set

Group of one or more *key codes* under a particular a name. A key set defines authorization within the framework of the authorization concept used (lock/key code concept or *access list* concept). A key set can be assigned to a *user ID*, an *LTERM partner* an (OSI) *LPAP partner*, a *service* or a *TAC queue*.

linkage program

See *KDCROOT*.

local secondary storage area

See *secondary storage area*.

Log4j

Log4j is part of the Apache Jakarta project. Log4j provides information for logging information (runtime information, trace records, etc.) and configuring the log output. *WS4UTM* uses the software product Log4j for trace and logging functionality.

lock code

Code protecting an LTERM partner or transaction code against unauthorized access. Access is only possible if the *key set* of the accesser contains the appropriate *key code* (lock/key code concept).

logging process

Process in Unix, Linux and Windows systems that controls the logging of account records or monitoring data.

LPAP bundle

LPAP bundles allow messages to be distributed to LPAP partners across several partner applications. If a UTM application has to exchange a very large number of messages with a partner application then load distribution may be improved by starting multiple instances of the partner application and distributing the messages across the individual instances. In an LPAP bundle, *open*UTM is responsible for distributing the messages to the partner application instances. An LPAP bundle consists of a master LPAP and multiple slave LPAPs. The slave LPAPs are assigned to the master LPAP on UTM generation. LPAP bundles exist for both the OSI TP protocol and the LU6.1 protocol.

LPAP partner

In the case of *distributed processing* via the *LU6.1* protocol, an LPAP partner for each partner application must be configured in the local application. The LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned LPAP partner and not by the application name or address.

LTERM bundle

An LTERM bundle (connection bundle) consists of a master LTERM and multiple slave LTERMs. An LTERM bundle (connection bundle) allows you to distribute queued messages to a logical partner application evenly across multiple parallel connections.

LTERM group

An LTERM group consists of one or more alias LTERMs, the group LTERMs and a primary LTERM. In an LTERM group, you assign multiple LTERMs to a connection.

LTERM partner

LTERM partners must be configured in the application if you want to connect clients or printers to a *UTM application*. A client or printer can only be connected if an LTERM partner with the appropriate properties is assigned to it. This assignment is generally made in the *configuration*, but can also be made dynamically using terminal pools.

LTERM pool

The TPOOL statement allows you to define a pool of LTERM partners instead of issuing one LTERM and one PTERM statement for each *client*. If a client establishes a connection via an LTERM pool, an LTERM partner is assigned to it dynamically from the pool.

LU6.1

Device-independent data exchange protocol (industrial standard) for transaction-oriented *server-server communication*.

LU6.1-LPAP bundle

LPAP bundle for *LU6.1* partner applications.

LU6.1 partner

Partner of the *UTM application* that communicates with the UTM application via the *LU6.1* protocol.

Examples of this type of partner are:

– a UTM application that communicates via LU6.1
– an application in the IBM environment (e.g. CICS, IMS or TXSeries) that communicates via LU6.1

main process (Unix systems / Windows systems)

Process which starts the *UTM application*. It starts the *work processes*, the *UTM system processes*, *printer processes, network processes, logging process* and the *timer process* and monitors the *UTM application*.

main routine KDCROOT

See *KDCROOT*.

management unit

SE Servers component; in combination with the *SE Manager*, permits centralized, web-based management of all the units of an SE server.

mapped host name
Mapping of the UTM host name to a real host name or vice versa.

message definition file
The message definition file is supplied with openUTM and, by default, contains the UTM message texts in German and English together with the definitions of the message properties. Users can take this file as a basis for their own message modules.

message destination
Output medium for a *message*. Possible message destinations for a message from the openUTM transaction monitor include, for instance, terminals, *TS applications*, the *event service* MSGTAC, the *system log file* SYSLOG or *TAC queues, asynchronous TACs, USER queues*, SYSOUT/SYSLST or stderr/stdout.
The message destinations for the messages of the UTM tools are SYSOUT/SYSLST and stderr/stdout.

message queue
Queue in which specific messages are kept with transaction management until further processed. A distinction is drawn between *service-controlled queues* and *UTM-controlled queues*, depending on who monitors further processing.

message queuing
Message queuing (MQ) is a form of communication in which the messages are exchanged via intermediate queues rather than directly. The sender and recipient can be separated in space or time. The transfer of the message is independent of whether a network connection is available at the time or not. In openUTM there are *UTM-controlled queues* and *service-controlled queues*.

MSGTAC
Special event service that processes messages with the message destination MSGTAC by means of a program. MSGTAC is an asynchronous service and is created by the operator of the application.

multiplex connection (BS2000 systems)
Special method offered by *OMNIS* to connect terminals to a *UTM application*. A multiplex connection enables several terminals to share a single transport connection.

multi-step service (KDCS)
Service carried out in a number of *dialog steps*.

multi-step transaction
Transaction which comprises more than one *processing step*.

Network File System/Service / NFS

Allows Unix systems to access file systems across the network.

network process (Unix systems / Windows systems)

A process in a *UTM application* for connection to the network.

network selector

The network selector identifies a service access point to the network layer of the *OSI reference model* in the local system.

node

Individual computer of a *cluster*.

node application

UTM application that is executed on an individual *node* as part of a *UTM cluster application*.

node bound service

A node bound service belonging to a user can only be continued at the node application at which the user was last signed on. The following services are always node bound:

– Services that have started communications with a job receiver via LU6.1 or OSI TP and for which the job-receiving service has not yet been terminated
– Inserted services in a service stack
– Services that have completed a SESAM transaction

In addition, a user's service is node bound as long as the user is signed-on at a node application.

node filebase

Filename prefix or directory name for the *node application's KDCFILE*, *user log file* and *system log file*.

node recovery

If a node application terminates abnormally and no rapid warm start of the application is possible on its associated *node computer* then it is possible to perform a node recovery for this node on another node in the UTM cluster. In this way, it is possible to release locks resulting from the failed node application in order to prevent unnecessary impairments to the running *UTM cluster application*.

normal termination of a UTM application

Controlled termination of a *UTM application*. Among other things, this means that the administration data in the *KDCFILE* are updated. The *administrator* initiates normal termination (e.g. with KDCSHUT N). After a normal termination, openUTM carries out any subsequent start as a *cold start*.

object identifier

An object identifier is an identifier for objects in an OSI environment which is unique throughout the world. An object identifier comprises a sequence of integers which represent a path in a tree structure.

OMNIS (BS2000 systems)

OMNIS is a “session manager” which lets you set up connections from one terminal to a number of partners in a network concurrently OMNIS also allows you to work with multiplex connections.

online import

In a *UTM cluster application*, online import refers to the import of application data from a normally terminated node application into a running node application.

online update

In a *UTM cluster application,* online update refers to a change to the application configuration or the application program or the use of a new UTM revision level while a *UTM cluster application* is running.

open terminal pool

Terminal pool which is not restricted to clients of a single computer or particular type. Any client for which no computer- or type-specific terminal pool has been generated can connect to this terminal pool.

OpenCPIC

Carrier system for UTM clients that use the *OSI TP* protocol.

OpenCPIC client

OSI TP partner application with the *OpenCPIC* carrier system.

openSM2

The openSM2 product line offers a consistent solution for the enterprise-wide performance management of server and storage systems. openSM2 offers the acquisition of monitoring data, online monitoring and offline evaluation.

openUTM application

See *UTM application*.

openUTM cluster

From the perspective of UPIC clients, **not** from the perspective of the server: Combination of several node applications of a UTM cluster application to form one logical application that is addressed via a common symbolic destination name.

openUTM-D

openUTM-D (openUTM distributed) is a component of openUTM which allows *distributed processing*. openUTM-D is an integral component of openUTM.

OSI-LPAP bundle

LPAP bundle for *OSI TP* partner applications.

OSI-LPAP partner

OSI-LPAP partners are the addresses of the *OSI TP partners* generated in openUTM. In the case of *distributed processing* via the *OSI TP* protocol, an OSI-LPAP partner for each partner application must be configured in the local application. The OSI-LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned OSI-LPAP partner and not by the application name or address.

OSI reference model

The OSI reference model provides a framework for standardizing communications in open systems. ISO, the International Organization for Standardization, described this model in the ISO IS7498 standard. The OSI reference model divides the necessary functions for system communication into seven logical layers. These layers have clearly defined interfaces to the neighboring layers.

OSI TP

Communication protocol for distributed transaction processing defined by ISO. OSI TP stands for Open System Interconnection Transaction Processing.

OSI TP partner

Partner of the UTM application that communicates with the UTM application via the OSI TP protocol.

Examples of such partners are:

– a UTM application that communicates via OSI TP
– an application in the IBM environment (e.g. CICS) that is connected via openUTM-LU62
– an application of the OpenCPIC carrier system of the openUTM client
– applications from other TP monitors that support OSI TP

outbound conversation (CPI-C)

See *outgoing conversation*.

outgoing conversation (CPI-C)

A conversation in which the local CPI-C program is the *initiator* is referred to as an outgoing conversation. In the X/Open specification, the term “outbound conversation” is used synonymously with “outgoing conversation”.

page pool

Part of the *KDCFILE* in which user data is stored.
In a *standalone application* this data consists, for example, of *dialog messages*, messages sent to *message queues*, *secondary memory areas*.
In a UTM cluster application, it consists, for example, of messages to *message queues, TLS*.

parameter area

Data structure in which a program unit passes the operands required for a UTM call to openUTM.

partner application

Partner of a UTM application during *distributed processing*. Higher communication protocols are used for distributed processing (*LU6.1*, *OSI TP* or LU6.2 via the openUTM-LU62 gateway).

postselection (BS2000 systems)

Selection of logged UTM events from the SAT logging file which are to be evaluated. Selection is carried out using the SATUT tool.

prepare to commit (PTC)

Specific state of a distributed transaction
Although the end of the distributed transaction has been initiated, the system waits for the partner to confirm the end of the transaction.

preselection (BS2000 systems)

Definition of the UTM events which are to be logged for the *SAT audit*. Preselection is carried out with the UTM-SAT administration functions. A distinction is made between event-specific, user-specific and job-specific (TAC-specific) preselection.

presentation selector

The presentation selector identifies a service access point to the presentation layer of the *OSI reference model* in the local system.

primary storage area

Area in main memory to which the *KDCS program unit* has direct access, e.g. *standard primary working area*, *communication area*.

print administration

Functions for *print control* and the administration of *queued output jobs*, sent to a printer.

print control

openUTM functions for controlling print output.

printer control LTERM

A printer control LTERM allows a client or terminal user to connect to a UTM application. The printers assigned to the printer control LTERM can then be administered from the client program or the terminal. No administration rights are required for these functions.

printer control terminal

This term has been superseded by *printer control LTERM*.

printer group (Unix systems)

For each printer, a Unix system sets up one printer group by default that contains this one printer only. It is also possible to assign several printers to one printer group or to assign one printer to several different printer groups.

printer pool

Several printers assigned to the same *LTERM partner*.

printer process (Unix systems)

Process set up by the *main process* for outputting *asynchronous messages* to a *printer group*. The process exists as long as the printer group is connected to the *UTM application*. One printer process exists for each connected printer group.

process

The openUTM manuals use the term “process” as a collective term for processes (Unix systems / Windows systems) and tasks (BS2000 systems).

processing step

A processing step starts with the receipt of a *dialog message* sent to the *UTM application* by a *client* or another server application. The processing step ends either when a response is sent, thus also terminating the *dialog step*, or when a dialog message is sent to a third party.

program interface for administration

UTM program interface which helps users to create their own *administration programs*. Among other things, the program interface for administration provides functions for *dynamic configuration*, for modifying properties and application parameters and for querying information on the configuration and the current workload of the application.

program space (BS2000 systems)
Virtual address space of BS2000 which is divided into memory classes and in which both executable programs and pure data are addressed.

program unit
UTM *services* are implemented in the form of one or more program units. The program units are components of the *application program*. Depending on the employed API, they may have to contain KDCS, XATMI or CPIC calls. They can be addressed using *transaction codes*. Several different transaction codes can be assigned to a single program unit.

queue
See *message queue*.

queued output job
Queued output jobs are *asynchronous jobs* which output a message, such as a document, to a printer, a terminal or a transport system application.
Queued output jobs are processed by UTM system functions exclusively, i.e. it is not necessary to create program units to process them.

Quick Start Kit
A sample application supplied with openUTM (Windows systems).

redelivery
Repeated delivery of an *asynchronous message* that could not be processed correctly because, for example, the *transaction* was rolled back or the *asynchronous service* was terminated abnormally. The message is returned to the message queue and can then be read and/or processed again.

reentrant program
Program whose code is not altered when it runs. On BS2000 systems this constitutes a prerequisite for using *shared code*.

request
Request from a *client* or another server for a *service function*.

requestor
In XATMI, the term requestor refers to an application which calls a service.

resource manager
Resource managers (RMs) manage data resources. Database systems are examples of resource managers. openUTM, however, also provides its own resource managers for accessing message queues, local memory areas and

logging files, for instance. Applications access RMs via special resource manager interfaces. In the case of database systems, this will generally be SQL and in the case of openUTM RMs, it is the KDCS interface.

restart

See *screen restart*,
see *service restart*.

RFC1006

A protocol defined by the IETF (Internet Engineering Task Force) belonging to the TCP/IP family that implements the ISO transport services (transport class 0) based on TCP/IP.

RSA

Abbreviation for the inventors of the RSA encryption method (Rivest, Shamir and Adleman). This method uses a pair of keys that consists of a public key and a private key. A message is encrypted using the public key, and this message can only be decrypted using the private key. The pair of RSA keys is created by the UTM application.

SAT audit (BS2000 systems)

Audit carried out by the SAT (Security Audit Trail) component of the BS2000 software product SECOS.

screen restart

If a *dialog service* is interrupted, openUTM again displays the *dialog message* of the last completed *transaction* on screen when the service restarts provided that the last transaction output a message on the screen.

SE manager

Web-based graphical user interface (GUI) for the SE series of Business Servers. SE Manager runs on the *management unit* and permits the central operation and administration of server units (with /390 architecture and/or x86 architecture), application units (x86 architecture), net unit and peripherals.

SE server

A Business Server from Fujitsu's SE series.

secondary storage area

Memory area secured by transaction logging and which can be accessed by the KDCS *program unit* with special calls. Local secondary storage areas (LSSBs) are assigned to one *service*. Global secondary storage areas (GSSBs) can be

accessed by all services in a *UTM application*. Other secondary storage areas include the *terminal-specific long-term storage (TLS)* and the *user-specific long-term storage (ULS)*.

selector

A selector identifies a service access point to services of one of the layers of the *OSI reference model* in the local system. Each selector is part of the address of the access point.

semaphore (Unix systems / Windows systems)

Unix systems and Windows systems resource used to control and synchronize processes.

server

A server is an *application* which provides *services*. The computer on which the applications are running is often also referred to as the server.

server-server communication

See *distributed processing*.

server side of a conversation (CPI-C)

This term has been superseded by *acceptor*.

service

Services process the *jobs* that are sent to a server application. A service of a UTM application comprises one or more transactions. The service is called with the *service TAC*. Services can be requested by *clients* or by other servers.

service access point

In the OSI reference model, a layer has access to the services of the layer below at the service access point. In the local system, the service access point is identified by a *selector*. During communication, the *UTM application* links up to a service access point. A connection is established between two service access points.

service chaining (KDCS)

When service chaining is used, a follow-up service is started without a *dialog message* specification after a *dialog service* has completed .

service-controlled queue

Message queue in which the calling and further processing of messages is controlled by *services*. A service must explicitly issue a KDCS call (DGET) to read the message. There are service-controlled queues in openUTM in the variants *USER queue*, *TAC queue* and *temporary queue*.

service restart (KDCS)

If a service is interrupted, e.g. as a result of a terminal user signing off or a *UTM application* being terminated, openUTM carries out a *service restart*. An *asynchronous service* is restarted or execution is continued at the most recent *synchronization point*, and a *dialog service* continues execution at the most recent *synchronization point*. As far as the terminal user is concerned, the service restart for a dialog service appears as a *screen restart* provided that a dialog message was sent to the terminal user at the last synchronization point.

service routine

See *program unit*.

service stacking (KDCS)

A terminal user can interrupt a running *dialog service* and insert a new dialog service. When the inserted *service* has completed, the interrupted service continues.

service TAC (KDCS)

Transaction code used to start a *service*.

session

Communication relationship between two addressable units in the network via the SNA protocol *LU6.1*.

session selector

The session selector identifies an *access point* in the local system to the services of the session layer of the *OSI reference model*.

shared code (BS2000 systems)

Code which can be shared by several different processes.

shared memory

Virtual memory area which can be accessed by several different processes simultaneously.

shared objects (Unix systems / Windows systems)

Parts of the *application program* can be created as shared objects. These objects are linked to the application dynamically and can be replaced during live operation. Shared objects are defined with the KDCDEF statement SHARED-OBJECT.

sign-on check

See *system access control*.

sign-on service (KDCS)

Special *dialog service* for a user in which *program units* control how a user signs on to a UTM application.

single-step service

Dialog service which encompasses precisely one *dialog step*.

single-step transaction

Transaction which encompasses precisely one *dialog step*.

SOA

(Service-Oriented Architecture)
SOA is a system architecture concept in which functions are implemented in the form of re-usable, technically independent, loosely coupled *services*. Services can be called independently of the underlying implementations via interfaces which may possess public and, consequently, trusted specifications. Service interaction is performed via a communication infrastructure made available for this purpose.

SOAP

SOAP (Simple Object Access Protocol) is a protocol used to exchange data between systems and run remote procedure calls. SOAP also makes use of the services provided by other standards, XML for the representation of the data and Internet transport and application layer protocols for message transfer.

socket connection

Transport system connection that uses the socket interface. The socket interface is a standard program interface for communication via TCP/IP.

standalone application

See *standalone UTM application*.

standalone UTM application

Traditional *UTM application* that is not part of a *UTM cluster application*.

standard primary working area (KDCS)

Area in main memory available to all KDCS *program units*. The contents of the area are either undefined or occupied with a fill character when the program unit starts execution.

start format

Format output to a terminal by openUTM when a user has successfully signed on to a *UTM application* (except after a *service restart* and during sign-on via the *sign-on service*).

static configuration

Definition of the *configuration* during generation using the UTM tool *KDCDEF*.

SYSLOG file

See *system log file*.

synchronization point, consistency point

The end of a *transaction*. At this time, all the changes made to the *application information* during the transaction are saved to prevent loss in the event of a crash and are made visible to others. Any locks set during the transaction are released.

system access control

A check carried out by openUTM to determine whether a certain *user ID* is authorized to work with the *UTM application*. The authorization check is not carried out if the UTM application was generated without user IDs.

system log file

File or file generation to which openUTM logs all UTM messages for which SYSLOG has been defined as the *message destination* during execution of a *UTM application*.

TAC

See *transaction code*.

TAC queue

Message queue generated explicitly by means of a KDCDEF statement. A TAC queue is a *service-controlled queue* that can be addressed from any service using the generated name.

temporary queue

Message queue created dynamically by means of a program that can be deleted again by means of a program (see *service-controlled queue*).

terminal-specific long-term storage (KDCS)

Secondary storage area assigned to an *LTERM, LPAP* or *OSI-PAP partner* and which is retained after the application has terminated.

time-driven job

Job which is buffered by openUTM in a *message queue* up to a specific time until it is sent to the recipient. The recipient can be an *asynchronous service* of the same application, a *TAC queue*, a partner application, a terminal or a printer. Time-driven jobs can only be issued by KDCS *program units*.

timer process (Unix systems / Windows systems)

Process which accepts jobs for controlling the time at which *work processes* are executed. It does this by entering them in a job list and releasing them for processing after a time period defined in the job list has elapsed.

TNS (Unix systems / Windows systems)

Abbreviation for the Transport Name Service. TNS assigns a transport selector and a transport system to an application name. The application can be reached through the transport system.

Tomcat

see *Apache Tomcat*

transaction

Processing section within a *service* for which adherence to the *ACID properties* is guaranteed. If, during the course of a transaction, changes are made to the *application information*, they are either made consistently and in their entirety or not at all (all-or-nothing rule). The end of the transaction forms a *synchronization point*.

transaction code/TAC

Name which can be used to identify a *program unit*. The transaction code is assigned to the program unit during *static* or *dynamic configuration*. It is also possible to assign more than one transaction code to a program unit.

transaction rate

Number of *transactions* successfully executed per unit of time.

transfer syntax

With *OSI TP*, the data to be transferred between two computer systems is converted from the local format into transfer syntax. Transfer syntax describes the data in a neutral format which can be interpreted by all the partners involved. An *Object Identifier* must be assigned to each transfer syntax.

transport selector

The transport selector identifies a service access point to the transport layer of the *OSI reference model* in the local system.

transport system application

Application which is based directly on a transport system interface (e.g. CMX, DCAM or socket). When transport system applications are connected, the partner type APPLI or SOCKET must be specified during *configuration*. A transport system application cannot be integrated in a *distributed transaction*.

TS application

See *transport system application.*

typed buffer (XATMI)

Buffer for exchanging typed and structured data between communication partners. Typed buffers ensure that the structure of the exchanged data is known to both partners implicitly.

UPIC

Carrier system for openUTM clients. UPIC stands for Universal Programming Interface for Communication.

UPIC Analyzer

Component used to analyze the UPIC communication recorded with *UPIC Capture*. This step is used to prepare the recording for playback using *UPIC Replay.*

UPIC Capture

Used to record communication between UPIC clients and UTM applications so that this can be replayed subsequently (*UPIC Replay*).

UPIC client

The designation for openUTM clients with the UPIC carrier system.

UPIC Replay

Component used to replay the UPIC communication recorded with *UPIC Capture* and prepared with *UPIC Analyzer.*

user exit

This term has been superseded by *event exit.*

user ID

Identifier for a user defined in the *configuration* for the *UTM application* (with an optional password for *system access control*) and to whom special data access rights (*system access control*) have been assigned. A terminal user must specify this ID (and any password which has been assigned) when signing on to the UTM application. On BS2000 systems, system access control is also possible via *Kerberos*.

For other clients, the specification of a user ID is optional, see also *connection user ID*.

UTM applications can also be generated without user IDs.

user log file

File or file generation to which users write variable-length records with the KDCS LPUT call. The data from the KB header of the *KDCS communication area* is prefixed to every record. The user log file is subject to transaction management by openUTM.

USER queue

Message queue made available to every user ID by openUTM. A USER queue is a *service-controlled queue* and is always assigned to the relevant user ID. You can restrict the access of other UTM users to your own USER queue.

user-specific long-term storage

Secondary storage area assigned to a *user ID*, a *session* or an *association* and which is retained after the application has terminated.

USLOG file

See *user log file.*

UTM application

A UTM application provides *services* which process jobs from *clients* or other applications. openUTM is responsible for transaction logging and for managing the communication and system resources. From a technical point of view, a UTM application is a process group which forms a logical server unit at runtime.

UTM cluster application

UTM application that has been generated for use on a cluster and that can be viewed logically as a **single** application.
In physical terms, a UTM cluster application is made up of several identically generated UTM applications running on the individual cluster *nodes*.

UTM cluster files

Blanket term for all the files that are required for the execution of a UTM cluster application. This includes the following files:

- *Cluster configuration file*
- *Cluster user file*
- Files belonging to the *cluster page pool*
- *Cluster GSSB file*
- *Cluster ULS file*
- Files belonging to the *cluster administration journal**
- *Cluster lock file**
- Lock file for start serialization* (only in Unix systems and Windows systems)

The files indicated by * are created when the first node application is started. All the other files are created on generation using KDCDEF.

UTM-controlled queue

Message queues in which the calling and further processing of messages is entirely under the control of openUTM. See also *asynchronous job, background job* and *asynchronous message*.

UTM-D

See *openUTM-D*.

UTM-F

UTM applications can be generated as UTM-F applications (UTM fast). In the case of UTM-F applications, input from and output to hard disk is avoided in order to increase performance. This affects input and output which *UTM-S* uses to save user data and transaction data. Only changes to the administration data are saved.
In UTM cluster applications that are generated as UTM-F applications (APPLI-MODE=FAST), application data that is valid throughout the cluster is also saved. In this case, GSSB and ULS data is treated in exactly the same way as in UTM cluster applications generated with UTM-S. However, service data relating to users with RESTART=YES is written only when the relevant user signs off and not at the end of each transaction.

UTM generation

Static configuration of a *UTM application* using the UTM tool KDCDEF and creation of an application program.

UTM message

Messages are issued to *UTM message destinations* by the openUTM transaction monitor or by UTM tools (such as *KDCDEF*). A message comprises a message number and a message text, which can contain *inserts* with current values. Depending on the message destination, either the entire message is output or only certain parts of the message, such as the inserts).

UTM page

A UTM page is a unit of storage with a size of either 2K, 4K or 8 K. In *standalone UTM applications*, the size of a UTM page on generation of the UTM application can be set to 2K, 4K or 8 K. The size of a UTM page in a *UTM cluster application* is always 4K or 8 K. The *page pool* and the restart area for the KDCFILE and *UTM cluster files* are divided into units of the size of a UTM page.

utmpath (Unix systems / Windows systems)

The directory under which the openUTM components are installed is referred to as *utmpath* in this manual.
To ensure that openUTM runs correctly, the environment variable UTMPATH must be set to the value of *utmpath*. On Unix and Linux systems, you must set UTMPATH before a UTM application is started. On Windows systems UTMPATH is set in accordance with the UTM version installed most recently.

UTM-S

In the case of UTM-S applications, openUTM saves all user data as well as the administration data beyond the end of an application and any system crash which may occur. In addition, UTM-S guarantees the security and consistency of the application data in the event of any malfunction. UTM applications are usually generated as UTM-S applications (UTM secure).

UTM SAT administration (BS2000 systems)

UTM-SAT administration functions control which UTM events relevant to security which occur during operation of a *UTM application* are to be logged by *SAT*. Special authorization is required for UTM-SAT administration.

UTM system process

UTM process that is started in addition to the processes specified via the start parameters and which only handles selected jobs. UTM system processes ensure that UTM applications continue to be reactive even under very high loads.

UTM terminal

This term has been superseded by *LTERM partner*.

virtual connection

Assignment of two communication partners.

warm start

Start of a *UTM-S* application after it has terminated abnormally. The *application information* is reset to the most recent consistent state. Interrupted *dialog services* are rolled back to the most recent *synchronization point*, allowing processing

to be resumed in a consistent state from this point (*service restart*). Interrupted *asynchronous services* are rolled back and restarted or restarted at the most recent *synchronization point*.
For *UTM-F* applications, only configuration data which has been dynamically changed is rolled back to the most recent consistent state after a restart due to a preceding abnormal termination.
In UTM cluster applications, th e global locks applied to GSSB and ULS on abnormal termination of this node application are released. In addition, users who were signed on at this node application when the abnormal termination occurred are signed off.

WebAdmin

Web-based tool for the administration of openUTM applications via a Web browser. WebAdmin includes not only the full function scope of the *administration program interface* but also additional functions.

Web service

Application which runs on a Web server and is (publicly) available via a standardized, programmable interface. Web services technology makes it possible to make UTM program units available for modern Web client applications independently of the programming language in which they were developed.

WinAdmin

Java-based tool for the administration of openUTM applications via a graphical user interface. WinAdmin includes not only the full function scope of the *administration program interface* but also additional functions.

work process (Unix systems / Windows systems)

A process within which the *services* of a *UTM application* run.

workload capture & replay

Family of programs used to simulate load situations; consisting of the main components *UPIC Capture*, *UPIC Analyzer* and *Upic Replay* and - on Unix, Linux and Windows systems - the utility program *kdcsort*. Workload Capture & Replay can be used to record UPIC sessions with UTM applications, analyze these and then play them back with modified load parameters.

WS4UTM

WS4UTM (**W**eb**S**ervices for open**UTM**) provides you with a convenient way of making a service of a UTM application available as a Web service.

XATMI

XATMI (X/Open Application Transaction Manager Interface) is a program interface standardized by X/Open for program-program communication in open networks.
The XATMI interface implemented in openUTM complies with X/Open's XATMI CAE Specification. The interface is available in COBOL and C. In openUTM, XATMI can communicate via the OSI TP, *LU6.1* and UPIC protocols.

XHCS (BS2000 systems)

XHCS (Extended Host Code Support) is a BS2000 software product providing support for international character sets.

XML

XML (eXtensible Markup Language) is a metalanguage standardized by the W3C (WWW Consortium) in which the interchange formats for data and the associated information can be defined.

Abbreviations

Please note: Some of the abbreviations used here derive from the German acronyms used in the original German product(s).

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AES	Advanced Encryption Standard
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder - Loader - Starter (BS2000 systems)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Coded Character Set
CCSN	Coded Character Set Name
CICS	Customer Information Control System
CID	Control Identification
CMX	Communication Manager in Unix, Linux and Windows Systems
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000 systems)
DB	Database
DC	Data Communication
DCAM	Data Communication Access Method

DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000 systems)
DMS	Data Management System
DNS	Domain Name Service
DP	Distribted Processing
DSS	Terminal (Datensichtstation)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans™
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GSSB	Global Secondary Storage Area
HIPLEX®	Highly Integrated System Complex (BS2000 systems)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IFG	Interactive Format Generator
ILCS	Inter-Language Communication Services (BS2000 systems)
IMS	Information Management System (IBM)
IPC	Inter-Process Communication
IRV	International Reference Version
ISO	International Organization for Standardization
Java EE	Java Platform, Enterprise Edition
JCA	Java EE Connector Architecture
JDK	Java Development Kit
KAA	KDCS Application Area
KB	Communication Area
KBPRG	KB Program Area
KDCADMI	KDC Administration Interface
KDCS	Compatible Data Communication Interface

KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000 systems)
LSSB	Local Secondary Storage Area
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000 systems)
NB	Message Area
NEA	Network Architecture for BS2000 Systems
NFS	Network File System/Service
NLS	Native Language Support
OLTP	Online Transaction Processing
OML	Object Module Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Process Identification
PIN	Personal Identification Number
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Computer Center Accounting Procedure
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption algorithm according to Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000 systems)
RTS	Runtime System
SAT	Security Audit Trail (BS2000 systems)
SECOS	Security Control System
SEM	SE Manager
SGML	Standard Generalized Markup Language
SLU	Secondary Logical Unit

SM2	Software Monitor 2
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primary Working Area
SQL	Structured Query Language
SSB	Secondary Storage Area
SSO	Single Sign-On
TAC	Transaction Code
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-Specific Long-Term Storage
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaction Mode)
TPR	Privileged Function State in BS2000 systems (Task Privileged)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Non-Privileged Function State in BS2000 systems (Task User)
TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universal Database System
UDT	Unstructured Data Transfer
ULS	User-Specific Long-Term Storage
UPIC	Universal Programming Interface for Communication
USP	UTM Socket Protocol
UTM	Universal Transaction Monitor
UTM-D	UTM Variant for Distributed Processing in BS2000 systems
UTM-F	UTM Fast Variant
UTM-S	UTM Secure Variant
UTM-XML	openUTM XML Interface

VGID	Service ID
VTSU	Virtual Terminal Support
WAN	Wide Area Network
WS4UTM	Web-Services for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (X/Open interface for acess to the resource manager)
XAP	X/OPEN ACSE/Presentation programming interface
XAP-TP	X/OPEN ACSE/Presentation programming interface Transaction Processing extension
XATMI	X/Open Application Transaction Manager Interface
XCS	Cross Coupled System
XHCS	eXtended Host Code Support
XML	eXtensible Markup Language

Related publications

You will find the manuals on the internet at *http://manuals.ts.fujitsu.com*. You can order printed copies of those manuals which are displayed with an order number.

PDF files of all openUTM manuals are included on the openUTM Enterprise DVD with open platforms and on the openUTM WinAdmin DVD (for BS2000 systems).

openUTM documentation

openUTM
Concepts and Functions
User Guide

openUTM
Programming Applications with KDCS for COBOL, C and C++
Core Manual

openUTM
Generating Applications
User Guide

openUTM
Using openUTM Applications on BS2000 Systems
User Guide

openUTM
Using openUTM Applications on Unix, Linux and Windows Systems
User Guide

openUTM
Administering Applications
User Guide

openUTM
Messages, Debugging and Diagnostics on BS2000 Systems
User Guide

openUTM
Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems
User Guide

openUTM
Creating Applications with X/Open Interfaces
User Guide

openUTM
XML for openUTM

openUTM Client (Unix systems)
for the OpenCPIC Carrier System
Client-Server Communication with openUTM
User Guide

openUTM Client
for the UPIC Carrier System
Client-Server Communication with openUTM
User Guide

openUTM WinAdmin
Graphical Administration Workstation for openUTM
Description and online help system

openUTM WebAdmin
Web Interface for Administering openUTM
Description and online help system

openUTM, openUTM-LU62
Distributed Transaction Processing
between openUTM and CICS, IMS and LU6.2 Applications
User Guide

openUTM (BS2000)
Programming Applications with KDCS for Assembler
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for Fortran
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for Pascal-XT
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for PL/I
Supplement to Core Manual

WS4UTM (Unix systems and Windows systems**)**
WebServices for openUTM

openUTM
Master Index

Documentation for the openSEAS product environment

BeanConnect
User Guide

JConnect
Connecting Java Clients to openUTM
User documentation and Java docs

WebTransactions
Concepts and Functions

WebTransactions
Template Language

WebTransactions
Web Access to openUTM Applications via UPIC

WebTransactions
Web Access to MVS Applications

WebTransactions
Web Access to OSD Applications

Documentation for the BS2000 environment

AID
Advanced Interactive Debugger
Core Manual
User Guide

AID
Advanced Interactive Debugger
Debugging of COBOL Programs
User Guide

AID
Advanced Interactive Debugger
Debugging of C/C++ Programs
User Guide

BCAM
BCAM Volume 1/2
User Guide

BINDER
User Guide

BS2000 OSD/BC
Executive Macros
User Guide

BLSSERV
Dynamic Binder Loader / Starter in BS2000/OSD
User Guide

DCAM
COBOL Calls
User Guide

DCAM
Macros
User Guide

DCAM
Program Interfaces
Description

FHS
Format Handling System for openUTM, TIAM, DCAM
User Guide

IFG for FHS
User Guide

HIPLEX AF
High-Availability of Applications in BS2000/OSD
Product Manual

HIPLEX MSCF
BS2000 Processor Networks
User Guide

IMON
Installation Monitor
User Guide

MT9750 (MS Windows)
9750 Emulation under Windows
Product Manual

OMNIS/OMNIS-MENU
Functions and Commands
User Guide

OMNIS/OMNIS-MENU
Administration and Programming
User Guide

OSS (BS2000)
OSI Session Service
User Guide

RSO
Remote SPOOL Output
User Guide

SECOS
Security Control System
User Guide

SECOS
Security Control System
Ready Reference

SESAM/SQL
Database Operation
User Guide

openSM2
Software Monitor
Volume 1: Administration and Operation

TIAM
User Guide

UDS/SQL
Database Operation
User Guide

Unicode in BS2000/OSD
Introduction

VTSU
Virtual Terminal Support
User Guide

XHCS
8-Bit Code and Unicode Support in BS2000/OSD
User Guide

Documentation for the Unix, Linux and Windows system environment

CMX V6.0 (Unix systems)
Betrieb und Administration (only available in German)
User Guide

CMX V6.0
Programming CMX Applications
Programming Guide

OSS (UNIX)
OSI Session Service
User Guide

PRIMECLUSTER*TM*
Concepts Guide (Solaris, Linux)

openSM2
The documentation of openSM2 is provided in the form of detailed online help systems, which are delivered with the product.

Other publications

XCPI-C (X/Open)
Distributed Transaction Processing
X/Open CAE Specification, Version 2
ISBN 1 85912 135 7

Reference Model Version 2 (X/Open)
Distributed Transaction Processing
X/Open Guide
ISBN 1 85912 019 9

TX (Transaction Demarcation) (X/Open)
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 094 6

XTAMI (X/Open)
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 130 6

XML
W3C specification (www consortium)
Web page: *http://www.w3.org/XML*

FUJITSU Software BS2000 BS2IDE
Eclipse-based Integrated Development Environment for BS2000
Web page: *https://bs2000.ts.fujitsu.com/bs2ide/*

Index

A

abstract syntax 446
abterm_services 605
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 366
access authorization
  certificate 229
access key
  shared memory segment 628, 631, 634, 638
access list
  LTAC 199
  TAC, TAC queue 220
access point
  for OSI TP connection 512
  local address 447
access privileges
  LU6.1 partner application 479
  OSI TP partner application 519
  user 578
  via LTERM partner 493
  via LTERM pool 565
access protection
  LTAC 490
  LTERM pool 494, 565
  transaction code 548, 550
access via ID card 578
access_list
  kc_ltac_str 491
  kc_tac_str 557
accesswait_sec 489
  kc_ltac_str (KC_MODIFY_OBJECT) 330
ACCOUNT 668
account 375, 611
accounting 611
  activating (KDCAPPL) 668
  activating (program) 375
  enable (program) 375
  parameters (data structure) 610
  weighting for LTAC 199, 491, 553
accounting phase
  administering 375
  administering (KDCAPPL) 668
  administering (program) 375
  UTM accounting 611
ACON 772
actcon
  kc_mux_str 507
  kc_tpool_str 568
active 515
  KC_MODIFY_OBJECT 338
adding to the configuration 73
  client 74, 207
  LTERM partner 74, 200
  printer 74, 207
  program unit 70, 78, 205
  terminal 74, 207
  transaction code 71, 78, 214
  TS application 76, 207
  UPIC client 76
  UPIC client program 207
  user ID 72, 79, 222
  VORGANG exit 70, 78, 205
address
  format/uniqueness 83
  local OSI TP access point 447
  LU6.1 partner application 462
  OSI TP partner application 511

address format
BCAMAPPL name 455
client address 537
LU6.1 partner address 194, 463
OSI TP access point 451
ADJTABLE.TXT 848
AdjTable.txt 848
ADM-CMD
SAT in BS2000 119
ADMI trace 614
enabling/disabling 369
admi_trace
KC_MODIFY_OBJECT 369
ADMI-DIAGAREA 119
ADMI-USERAREA 119
admin 550
KC_CREATE_OBJECT 216
administer
LTERM pool (KDCPOOL) 762
LTERM pool (program) 355
session (KDCLSES) 751
session (program) 328
several applications simultaneously 133
administration
automatic 147
central 121
central, using commands 140
in dialog (commands) 110
local printers 802
message queues (DADM) 805
printers (KDCPADM) 830
printers (PADM) 800, 812
using message queuing (command) 112
UTM cluster application 122
via commands 109
via distributed processing 133
via TS application 138
via UTM client 128
with WinAdmin 123
administration authorization 153
administration call
modifying output destination (program) 379
several 116
administration command
asynchronous, receiver 630, 649
entering 110
format 665
generating 34, 157
KDCAPPL 666
KDCBNDL 680
KDCDIAG 681
KDCHELP 690
KDCINF 691
KDCLOG 742
KDCLPAP 743
KDCLSES 751
KDCLTAC 753
KDCLTERM 755
KDCMUX 758
KDCPOOL 762
KDCPROG 764
KDCPTERM 769
KDCSEND 774
KDCSHUT 775
KDCSLOG 779
KDCSWTCH 785
KDCTAC 789
KDCTCL 792
KDCUSER 797
output of results 110
sending to UTM 140
administration functions
KDCADM, overview 35
KDCADMI, overview 38
administration interface 31
diagnosis 119
administration journal 886
administration privileges
applications with user ID 156
applications without user IDs 156
for transaction code 550
granting 156
LU6.1 partner application 227, 479
OSI TP partner application 520
printers 802
user ID 583

administration program
as a message interface 144
central 144
decentralized 141
generating 154
KDCADM 109
multi-step service 117
portable 142
structure 115
writing 115
administration tool 31
administration tool CALLUTM 42, 853
administrator message
KC_SEND_MESSAGE 404
KDCSEND 774
AEQ
OSI TP access point 448
OSI TP partner application 518
AES encryption 268
alias name 861
all information
KDCINF 694
altlib 477
annoamsg 497
kc_lterm_str 497
kc_lterm_str (KC_CREATE_OBJECT) 203
kc_tpool_str 567
ap_name 447
api 553
KC_CREATE_OBJECT 218
APPLI client
creating, example 77
dynamic creation 76
dynamic deletion 87
entering dynamically 76
APPLI name
local application 602, 627, 649
application
terminating (KDCSHUT) 775
terminating (program) 413
with distributed processing 650
with TAC classes 649
application context 453
OSI TP partner application 518
application data
importing online 398
application entity qualifier
OSI TP access point 448
OSI TP partner application 518
application name
for heterogeneous link 479
application parameters
current values (data structure) 600
data structures 591
information about (KDCINF) 697
information about (program) 278, 288
modifying (KDCAPPL) 666
modifying (program) 311, 317
application process title
local application 661
OSI TP partner application 518
application program
file generations currently loaded 607
replacing 61
replacing (KDCAPPL) 672
replacing (program) 179
application start
date, time 602
application_context 518
application_entity_qualifier
kc_access_point_str 448
kc_osi_lpap_str 518
application_process_title
kc_osi_lpap_str 518
kc_utmd_par_str 661
applimode
kc_curr_par_str 602
kc_max_par_str 627
kc_system_par_str 649
appliname
kc_curr_par_str 602
kc_max_par_str 627
kc_system_par_str 649
APT
local application 661
OSI TP partner application 518

ASCII/EBCDIC conversion
kc_lpap_str 481
kc_osi_con_str 515
kc_pterm_str 535, 568
ass_kset 522
assignment of TAC to program unit
information about (KDCINF) 697
association 519
information about (KDCINF) 699
information about (program) 287, 295
properties (data structure) 509
statistical information 510
status 510
timer for assignment 489
timer for idle state (KDCLPAP) 748
timer for idle state (program) 342
timer for occupation 198
association ID
outputting (KDCINF) 715
outputting (program) 289
association_id 509
association_names 518
asyn_services 586
asyn_ta_per_100sec 604
asynchronous commands
calling 112
for administration 140
output destination 140, 630, 649
summary 35
asynchronous jobs
administering (KDCDADM) 821
buffering (LTERM partner) 496
deleting (DADM) 810
deleting (KDCDADM) 822
identifying 807
information about (KDCDADM) 824
number currently stored 605
number of active 639
prioritizing (DADM) 809
prioritizing (KDCDADM) 829
asynchronous messages
administering (KDCDADM) 820
announcing (LTERM partner) 497
announcing (LTERM pool) 567
asynchronous messages (cont.)
current number (LPAP partner) 482
current number (LTERM partner) 501
current number (TAC) 554
maximum number (LTERM pool) 567
maximum number (OSI-LPAP partner) 521
maximum number (TAC) 551
asynchronous processing
determining number of processes (KDCAPPL) 670
determining number of processes (program) 380
maximum number of processes 653
number of processes 603
number of processes set 654
asynchronous service
importing online 398
asynchronous TAC
online import 398
asynchronous transaction code 551
asyntasks
kc_max_par_str 627
kc_tasks_par_str 653
atac_redelivery 639
authorization concept 153
authorization levels
administration privileges 153
authorization, administration 153
authorizations concept
printer administration 802
auto_connect
kc_con_str 464
kc_lpap_str 480
kc_lpap_str (KC_MODIFY_OBJECT) 325
kc_mux_str 506
kc_pterm_str 534
kc_pterm_str (KC_CREATE_OBJECT) 209
kc_pterm_str (KC_MODIFY_OBJECT) 344
auto_connect_number 519
KC_MODIFY_OBJECT 340
autolink function (BLS) 477, 707
automatic
administration 147
diagnostics 150

automatic connection setup
at start ,OSI TP partner 519
at start (KDCPTERM) 772
at start (program) 209, 325, 344
at start, client/printer 534
at start, LU6.1 partner 464, 480
at start, multiplex connection 506
LPAP/OSI-LPAP (KDCLPAP) 747
multiplex connection (KDCMUX) 759
multiplex connection (program) 336
OSI-LPAP partner (KDCINF) 717
OSI-LPAP partner (program) 339
to printer 496
automatic KDCSIGN 75
automatic mode, print output 816
automatic restart
terminal/client 497
user ID 582
automatic service restart 203
automatic size monitoring
SYSLOG 637, 780
availability
dynamically created objects 74
avg_cpgpool_size 362
avg_pool_size 605
kc_curr_par_str
(KC_MODIFY_OBJECT) 365
avg_saved_pgs_by_compr 609
avg_wait_time_msec 560

B

base name
KDCFILE 632
base_gen 432
bc_name 454
BCAM, passing messages 628
BCAM trace
activate/deactivate (KDCDIAG) 685
activate/deactivate (program) 327, 334, 369
enabling/disabling (program) 369, 370, 371, 372
general 612
LPAP-specific 482
LTERM-specific 500
bcam_trace
kc_diag_and_account_par_str 327, 369, 612
kc_lpap_str 482
kc_lterm_str 500
kc_lterm_str (KC_MODIFY_OBJECT) 334
kc_mux_str 508
kc_user_str 588
kc_user_str (KC_MODIFY_OBJECT) 361
bcamappl
kc_con_str 463
kc_lses_str 485
kc_lses_str (KC_MODIFY_OBJECT) 328
kc_lterm_str 499
kc_mux_str 506
kc_pterm_str 531
kc_pterm_str (KC_CREATE_OBJECT) 208
kc_subnet_str 546
kc_tpool_str 564
big endian 142
blksize 627
bound_ptc 590
bound_service 590
bretrynr 628
kc_max_par_str 377
BTRACE 369, 685
bundle 483, 500, 522
byte representation 142

C

cache
controlling paging (KDCAPPL) 668
controlling paging (program) 377
current data about 365, 605
paging 628
resident 628
size 628
cache_hit_rate 605
kc_curr_par_str
(KC_MODIFY_OBJECT) 365
cache_location
kc_max_par_str 639
cache_wait_buffer 605
kc_curr_par_str
(KC_MODIFY_OBJECT) 366

cacheshmkey 628
cachesize_pages 628
cachesize_paging 628
KC_MODIFY_OBJECT 377
cachesize_res 628
CALC 668
calc 611
KC_MODIFY_OBJECT 375
calculation phase
activating/deactivating (program) 375
enable/disable (KDCAPPL) 668
UTM accounting 611
call_type 551
KC_CREATE_OBJECT 217
CALLUTM 42, 853
assign syntax file 876
components 876
integration in a UTM application 876
KDCDEF generation 856
messages 878
program statements 860
program-monitoring job variable 877
starting 858
system environment 876
CALLUTM statement
CONTINUE-SERVICE 863
CREATE-CONFIGURATION 866
DEALLOCATE-CONVERSATION 869
MODIFY-CONFIGURATION 870
notational conventions 861
SELECT-SERVICE 871
SHOW-CONFIGURATION 875
cancel
asynchronous jobs (KDCDADM) 822
messages (KDCDADM) 822
card information
length 629
magnetic stripe card 578
card_free 622
card_id 580
card_position 578
KC_CREATE_OBJECT 223
card_string 579
KC_CREATE_OBJECT 223
card_string_lth 578
KC_CREATE_OBJECT 223
card_string_type 579
KC_CREATE_OBJECT 223
card_total 622
cardlth 629
catalog ID, node application 459
catalog ID KDCFILE 629
catid_a 629
node application 459
catid_b 629
node application 459
CCS name
application 632
edit profile 471
LTERM partner 493
LTERM pool 565
user ID 581
ccsname 471
central administration 121
SDF command interface 128
transaction management 133
via distributed processing 133
via TS application 138
via UTM client 128
central administration programs 144
cert_auth
kc_user_str 588
certificate
at sign-on 229
kc_user_str 588
certificate_gen
kc_system_par_str 651
certification point 229
change_necessary 477
changeable 477
character set, extended
application 632
edit profile 471
locale, LTERM partner 493
locale, LTERM pool 565
user ID 581
cid 535
KC_CREATE_OBJECT 209

clear character, KB/SPAB 629
clear session
KC_MODIFY_OBJECT 329
KDCLSES 751
client
active (LTERM pool) 568
adding dynamically 74, 188, 207
availability 66
changing assignment (KDCSWTCH) 785
changing assignment (program) 343
computer name 499
defining the maximum number 55
deleting dynamically 87, 259
disabling 65
information about (program) 296
maximum number in configuration 620
modifying dynamically 96
number of messages exchanged 538
number that can be added dynamically 620
properties (data structure) 529
statistical information 538
TS application 534
client connection
duration 538
status 537
client, APPLI type
dynamic creation 76
dynamic deletion 87
client, UPIC type
dynamic deletion 87
clients_signed 587
clients, number of
changing for LTERM pool (KDCPOOL) 762
changing for LTERM pool (program) 355
clrch 629
clrch_type 629
cluster
administration 122
properties of the node applications (data structure) 457
cluster administration journal 886
cluster configuration file
file/directory name 594
cluster page pool 62
number of files 599
utilization 592
cluster user file
releasing lock 306
cluster_appl 651
CMX application
for administration 138
co_deleted
kc_con_str 465
co_name 463
COBUSER 41, 844
code type for RTAC name 197, 488
code_type 488
command format 665
command interface
overview 34
commands
central administration using 140
entering, in dialog 110
common memory pool
wait time 633
communication area
clear character 629
length 631
comp 523
kc_insert_str 618
compatibility guarantee 41
compiler
KC_CREATE_OBJECT 205
program unit/VORGANG exit 523
complexity level
password 582
components of CALLUTM 876
computer name
client/printer 499, 530
local computer 649
LTERM pool 563
LU6.1 partner 463
message router 506
CON
information about (KDCINF) 698
information about (program) 287

con 485
KC_MODIFY_OBJECT 328
con_freel
kc_dyn_par_str 623
con_total
kc_dyn_par_str 623
conbad
kc_con_str 465
kc_pterm_str 538
CONCTIME 669
conctime1_sec 659
KC_MODIFY_OBJECT 384
conctime2_sec 660
KC_MODIFY_OBJECT 384
concurrent_terminal_signon 644
configuration
dynamic changing 67
expanding dynamically 188
modifying dynamically 95
configuration entries
administration commands 157
configuration models
central administration 127
confirmation mode, print output 816
changing (KDCPADM) 835, 837
changing (PADM) 817
conn_users 630
KC_MODIFY_OBJECT 378
connect_mode
kc_con_str 464
kc_lpap_str 482
kc_lpap_str (KC_MODIFY_OBJECT) 326
kc_lses_str 485
kc_lses_str (KC_MODIFY_OBJECT) 328, 329
kc_lterm_str 500
kc_lterm_str (KC_MODIFY_OBJECT) 332
kc_mux_str 507
kc_pterm_str 537
kc_pterm_str (KC_MODIFY_OBJECT) 345
kc_tpool_str 564
kc_user_str 584
connect_number 521
KC_MODIFY_OBJECT 341
connect_state 510
connected_users 604
connection
avoiding bottlenecks 56
current status (LU6.1) 464
duration (client/printer) 538
duration (LU6.1) 464
duration (OSI TP) 510
establishing (KDCPTERM) 771
establishing (program) 332
establishing/shutting down (KDCLPAP) 745
for OSI-LPAP partner 521
information about (KDCINF) 699
information about (program) 287, 293
loss of (client/printer) 538
loss of (LU6.1) 465
number of messages (LU6.1) 464
number of parallel associations (OSI TP) 519
re-establishing (KDCAPPL) 669
re-establishing (program) 382
status (client) 537
status QUIET (LU6.1) 482
to client/printer, status 500
to LU6.1 partner, properties 462
to message router, status 507
to printer, terminating (PADM) 814
to printer, terminating(KDCPADM) 837
trace for (KDCDIAG) 685
trace for (program) 369
connection bundle
exchange master LTERM 680
connection request time 382, 657, 669
connection setup
automatic, at start (KDCPTERM) 772
automatic, at start (program) 344
automatic, to client/printer 534
automatic, to LU6.1 partner 464, 480
automatic, to multiplex connection 506
automatic, to OSI TP partner 519
automatic, to printer 496
global monitoring time 384, 659, 660
to printers (KDCPTERM) 771
to printers (program) 418

connection shutdown
property QUIET (LU6.1) 482
status QUIET (OSI TP) 521
connection user ID 76
CONRTIME 669
conrtime_min 657
KC_MODIFY_OBJECT 382
contention loser
LU6.1 session 489
OSI TP association 509
OSI TP connection 519
contention winner
LU6.1 session 480
OSI TP association 509
OSI TP connection 519
contime_min
kc_con_str 464
kc_osi_association_str 510
kc_pterm_str 538
CONTINUE-SERVICE 863
control
KC_GET_OBJECT output 279
KDCINF output 701
control statements
generating 101
control value
message queue (LTERM partner) 202, 496
message queue (LTERM pool) 567
message queue (OSI-LPAP) 521
message queue (TAC) 551
printer queue 496
contwin
kc_lpap_str 480
kc_osi_association_str 509
kc_osi_lpap_str 519
conversion
ASCII/EBCDIC 535, 568
parameter area 146
converting ASCII/EBCDIC 481, 515
COPY members
COBOL programming interface 840
cpgpool 599
cpgpool_fs 599
cpgpool_warnlevel 599
CPI-C
trace 370
CPI-C trace 615
enabling/disabling 370
cpic_trace
KC_MODIFY_OBJECT 370
CPU time
average (TAC) 349, 555
maximum, for program run 552
cpu_time_msec 552
KC_CREATE_OBJECT 218
cputime_sec 585
create
client 74, 207
KDCDEF control statements 246
KDCDEF input 252
KDCFILE (inverse KDCDEF) 252
key set 79
LTAC 81
LTERM partner 74, 195, 196
LU6.1 connection 80
LU6.1 session 80
new objects 73
printer 74, 207
program unit 70, 78, 205
terminal 207
transaction code 71, 78, 214
TS application 76, 207
user ID 72, 79
user ID for TS application 76
VORGANG exit 70, 78, 205
CREATE-CONFIGURATION 866
CSECT name
program unit 523
cterm 495
KC_CREATE_OBJECT 201
curr_asyntasks
kc_curr_par_str 603
kc_tasks_par_str 655
curr_date_... 603
curr_encryption 541
curr_gen 432
curr_jr 606

curr_max_asyntasks
kc_curr_par_str 603
kc_tasks_par_str 654
curr_max_tasks_in_pgwt
kc_curr_par_str 603
kc_tasks_par_str 654
curr_pool_size 604
curr_size_kbyte 430
curr_size_percent 430
curr_size_utmpages 430
curr_system_tasks 609
kc_tasks_par_str 655
curr_tasks
kc_curr_par_str 603
kc_tasks_par_str 655
curr_tasks_in_pgwt
kc_curr_par_str 603
kc_tasks_par_str 655
curr_time_... 603
current data
cache 365, 605
user ID 585
current date 603
current time 603
current values, application parameters 600

D

DADM
functions 799
program unit 820
reading user information 809
sample programs 820
DADM CS
prioritizing asynchronous jobs 809
prioritizing messages 809
DADM DA/DL
deleting asynchronous jobs 810
deleting messages 810
DADM RQ
information about message queues 808
data access control
LTAC 199
USER queue 228
via access list 158
data area 173
length 170
maximum size 173
data compression
enabling/disabling 57, 61
data global to the cluster
cluster page pool 62
data local to the node
UTM cluster application 62
data structure
application parameters 591
format of data 444
general structure 444
kc_abstract_syntax_str 446
kc_access_point_str 447
KC_ADM_RETCODE 174
kc_application_context_str 453
kc_bcamappl_str 454
kc_cluster_curr_par_str 122, 592
kc_cluster_node_str 122, 457
kc_cluster_par_str 122, 593
kc_con_str 462
kc_curr_par_str 600
kc_db_info_str 466
kc_diag_and_account_par_str 610
kc_dump_event_str 373, 374, 610
kc_dyn_par_str 619
kc_edit_str 468
kc_encrypt_advanced_str 276
kc_encrypt_str 277
kc_gssb_str 472
kc_insert_str 373, 374, 610, 617
kc_kset_str 473
kc_load_module_str 475
kc_lock_mgmt_str 309
kc_lpap_str 478
kc_lses_str 484
kc_ltac_str 487
kc_lterm_str 492
kc_max_par_str 624
kc_message_module_str 503
kc_msg_des_all_par_str 640
kc_mux_str 505
kc_online_import_str 400

data structure (cont.)
kc_osi_association_str 509
kc_osi_con_str 511
kc_osi_lpap_str 517
kc_program_str 523
kc_pterm_str 529
kc_queue_par_str 643
kc_queue_str 543
kc_sfunc_str 544
kc_shutdown_str 415
kc_syslog_str 430
kc_system_par_str 648
kc_tac_str 547
kc_tacclass_str 559
kc_tasks_par_str 652
kc_timer_par_str 656
kc_tpool_str 562
kc_transfer_syntax_str 572
kc_triple_str 171
kc_user_dyn1_str 288, 573
kc_user_dyn2_str 288, 573
kc_user_fix_str 288, 573
kc_user_str 288, 573
kc_utmd_par_str 661
KDCADMI 443
object properties 445
return code 174
data_compression
kc_curr_par_str 609
kc_max_par_str 639
data_lth 170
data_lth_ret 170
DATA-COMPRESSION 670
database calls
average number of (TAC) 348, 554
average processing time (TAC) 349, 555
database key 218, 552
databases
information 466
db_counter 554
db_elap_msec 555
db_entry_name 466
db_id 466
db_lib_info 467
db_type 466
db_xaswitch 467
dbkey 552
KC_CREATE_OBJECT 218
DCAM application 533
dead letter queue
move messages (DADM) 811
move messages (KDCDADM) 827
number of messages 639
dead_letter_q
kc_tac_str 558
dead_letter_q_alarm 639
deadlocks 606
kc_curr_par_str
(KC_MODIFY_OBJECT) 366
DEALLOCATE-CONVERSATION 869
decentralized administration programs 141
delayed delete 85
delete
key set 93
LTAC 94
LU6.1 connection 93
LU6.1 session 93
deleted
kc_ltac_str 491
kc_lterm_str 501
kc_program_str 526
kc_pterm_str 538
kc_tac_str 555
kc_user_str 585
deleted program units
information about (program) 294, 296
deleting
asynchronous jobs (KDCDADM) 822
messages (KDCDADM) 822
deleting from the configuration 85
client/printer 87, 259
effects 88, 90, 92
LTERM partner 87, 259
program unit 89, 259
transaction code 89, 259
user ID 91, 259
VORGANG exit 89
DES encryption 268

DESTADM 114, 140
for distributed processing 140
for TS application 140
modifying (program) 379
destadm
kc_max_par_str 379, 630
kc_system_par_str 649
dget_redelivery 639
diagnosis
administration interface 119
automatic 150
diagnostic aids
activating/deactivating (KDCDIAG) 681
activating/deactivating (program) 318
diagnostic dump
creating (program) 373
diagnostic information 638
diagnostic parameters
data structure 610
dial_conv_pages 642
dial_step_per_100sec 604
dial_ta_per_100sec 604
dialog commands
entering 110
output of results 110
overview 35
dialog TAC 551
disable
client/printer 534
LPAP partner 464, 480
LPAP partner (KDCLPAP) 748
LPAP partner (program) 325
LTAC 489
LTAC (KDCLTAC) 753
LTAC (program) 331
LTERM partner 494
LTERM partner (KDCLTERM) 756
LTERM partner (program) 332
LTERM partners in a pool 566
multiplex connection 507
OSI-LPAP (KDCLPAP) 748
OSI-LPAP partner 520
OSI-LPAP partner (program) 339
transaction code 549
disable (cont.)
transaction code (KDCTAC) 789
transaction code (program) 347
user ID 578
user ID (KDCUSER) 798
user ID (program) 357
distributed processing
administration via 133
application with 650
parameters (data structure) 661
statistical information 606
documentation
summary 16
dpn 481
DPUT ID, see job ID
DPUT jobs
execution time 630
number currently waiting 605
dputlimit1_... 630
dputlimit2_... 631
dump
creating 682
creating (program) 373
creating during operation (KDCDIAG) 682
creating during operation (program) 186
dump_event
kc_dump_event_str 616
dump_msg_id 612
duration of password validity 582
dynamic objects
information (data structure) 619
dynamically created objects
access to 73
availability 74
dynamically deleted objects
access to 85

E

EBCDIC 142
EBCDIC/ASCII conversion
kc_lpap_str 481
kc_osi_con_str 515
kc_pterm_str 535, 568
ed_name 469

edit profile options, data structure 468
edit_bell 469
edit_mode 469
enable
  LPAP partner (KDCLPAP) 748
  LPAP partner (program) 325
  LTAC (KDCLTAC) 753
  LTAC (program) 331
  LTERM partner (KDCLTERM) 756
  LTERM partner (program) 332
  OSI-LPAP (program) 339
  OSI-LPAP partner (KDCLPAP) 748
enable Stxit logging (program) 376
enable/disable trace
  BCAM (program) 369, 370, 371, 372
ENCRADM 846
encryption
  generate key 271
encryption_level 556
  kc_perm_str (KC_CREATE_OBJECT) 211
  kc_pterm_str 539, 569
entering in the configuration
  administration commands 34
  administration program 154
  LTERM partner 195, 196
  TS application 76
ENTRY name
  program unit 523
error
  formatting 819
establishing a connection
  global monitoring time 659
establishing a session
  global monitoring time 659
  KDCLSES 751
  program 328
event
  kc_dump_event_str 617
event_type
  kc_dump_event_str 616
example
  KDCTCL 796
execution time
  time-controlled job 630, 631
exit_name
  KC_CREATE_OBJECT 217
extend KDCFILE (inverse KDCDEF) 252
extended character set 581

F

fault handling
  print output 819
fgg 431
file generation
  switching, SYSLOG file (KDCSLOG) 780
  switching, SYSLOG file (program) 426
file_name 430
first_valid_gen 432
floating point presentation
  KDCINF 692
follow-up TAC 551
format
  administration command 665
  data structure description 445
  in the administration program 117
  object names 84
format indicator
  T-selector BCAMAPPL name 456
  T-selector of partner address 194
  T-selector OSI TP access point 452
  T-selector partner address 464, 516, 537
format_attr
  kc_lterm_str 495
  kc_lterm_str (KC_CREATE_OBJECT) 202
  kc_lterm_str (KC_MODIFY_OBJECT) 333
  kc_tpool_str 566
  kc_user_str 581
  kc_user_str (KC_CREATE_OBJECT) 225
  kc_user_str (KC_MODIFY_OBJECT) 361
format_name
  kc_lterm_str 495
  kc_lterm_str (KC_CREATE_OBJECT) 202
  kc_lterm_str (KC_MODIFY_OBJECT) 333
  kc_tpool_str 566
  kc_user_str 581
  kc_user_str (KC_CREATE_OBJECT) 225
  kc_user_str (KC_MODIFY_OBJECT) 361
formatting error, print output 819

fpmm_pages 642
fput_pages 642
free_pages 641
function call KDCADMI 161
function key
properties (data structure) 544

G

gen_system_tasks
kc_tasks_par_str 655
generate
UTM client for administration (cluster) 131
generated processes
maximum number 653
generation
administration commands 34, 157
administration program 154
CALLUTM 854
distributed processing via LU6.1 136
TS application 138
UTM client for administration 130
generation variant 602, 649
global secondary storage area
maximum number 631
name 472
global semaphore
privileges key 636
grace 644
Grace-Sign-On 644
gs_name 472
GSSB
maximum number 631
name 472
gssb_pages 641
gssbs 631

H

hardware fault during print output 819
hcopy 469
header file kcadminc.h 443
heterogeneous link
application name 479
HNDLUSR 844
hom 469
hostname
kc_cluster_node_str 458
kc_max_par_str 631
kc_system_par_str 649
HP-UX 15

I

ID of association 715
id_lth 170
identification
message queues 807
identification area 171
length 170
identification card 578
identify
asynchronous jobs 807
messages 807
idle state
association (KDCLPAP) 748
association (program) 342
OSI TP association, timer 520
session (KDCLPAP) 748
session (program) 326
timer, LU6.1 session 481
idle time
LU6.1 session 481
timer, OSI TP association 520
IDLETIME 748
idletime 539
kc_perm_str (KC_CREATE_OBJECT) 211
kc_tpool_str 569
idletime_sec
kc_lpap_str 481
kc_lpap_str (KC_MODIFY_OBJECT) 326
kc_osi_lpap_str 342, 520
kc_pterm_str (KC_MODIFY_OBJECT) 346, 356
ihdr 470
immediate delete 85
import_node 400
in_queue 554
in_queue_ex 558
in_service 585
incnt 507

incounter 501
indication_calls 510
information
  on administration commands 690
  select (KDCADMI) 172
information functions 46
insert_index
  kc_insert_str 617
INTEGER 197, 488
inverse KDCDEF 101, 246
  result 105
  starting 103
  version migrations 106
inverse_kdcdef_state 650
IP address
  client 540
  client (IPv6) 542
  LU6.1 partner 465
  LU6.1 partner (IPv6) 465, 516
  OSI TP partner 516
ip_addr
  kc_con_str 465
  kc_osi_con_str 516
  kc_pterm_str 540
ip_addr_format 546
ip_addr_v6
  kc_con_str 465
  kc_osi_con_str 516
  kc_pterm_str 542
ip_v
  kc_con_str 465, 516
  kc_pterm_str 542
ipc 631
IPC trace area 631
ipcshmkey 631
ipctrace 631
IPv4 subnet address
  kc_subnet_str 546
ipv4_address 546
IPv6 subnet address
  kc_subnet_str 546
ipv6_address 546
ISO code 142

J
job ID 807
job receiver
  central administration 133
job receiver confirmation
  wait time 660
job submitter
  using LU6.1 session 485
job-receiving services
  maximum number, remote 661

K
K009 545
K041 148
K091 148
K119 150
kaashmkey 631
KB
  clear character 629
  length 631
kb 631
KC_ABORT_PTC_SERVICE 306
kc_abstract_syntax_str 446
kc_access_point_str 447
kc_adm_parameter 162
KC_ADM_RETCODE 174
kc_application_context_str 453
KC_APPLICATION_PAR 285
KC_ASCENDING 285
KC_ATTRIBUTES 279, 285
KC_ATTRIBUTES_NEXT 279, 285
kc_bcamappl_str 454
KC_CHANGE_APPLICATION 179
  data returned 185
  return code 184
kc_change_application_str 185
KC_CLUSTER_CURR_PAR 288
kc_cluster_curr_par_str 122, 592
kc_cluster_node_str 122, 457
KC_CLUSTER_PAR 288
kc_cluster_par_str 122, 593
KC_CON 192
  return codes 231, 263
KC_CON_STMT 251

kc_con_str 193, 462
KC_CREATE_DUMP 186
KC_CREATE_OBJECT 188
  KC_CON 193
  KC_KSET 195
  KC_LSES 196
  KC_LTAC 197
  return codes 230
KC_CREATE_STATEMENTS 246
  return codes 192, 254
kc_create_statements_str 250
kc_curr_par_str 600
KC_DB_INFO 287
kc_db_info_str 466
KC_DELETE_OBJECT 256
  return codes 262
KC_DESCENDING 279, 285
KC_DEVICE_STMT 251
kc_diag_and_account_par_str 610
kc_dump_event_str 373, 374, 610
kc_dyn_par_str 619
kc_edit_str 468
KC_ENCRYPT
  return codes 275
kc_encrypt_advanced_str 276
kc_encrypt_str 276, 277
KC_GET_OBJECT 278
  return codes 298
kc_gssb_str 472
kc_insert_str 373, 374, 610, 617
KC_KSET 192, 287
  return codes 232, 263
KC_KSET_STMT 251
kc_kset_str 473
kc_load_module_str 475
kc_lock_mgmt_str 309
KC_LOCL_MGMT 306
kc_lpap_str 478
KC_LSES 192
  return codes 232, 263
kc_lses_str 196, 484
KC_LTAC 192
  return codes 232
kc_ltac_str 197, 487
KC_LTERM 192
  return codes 233, 264
kc_lterm_str 195, 492
kc_max_par_str 624
kc_message_module_str 503
KC_MODIFY_OBJECT 311, 610, 656
  KC_CLUSTER_NODE 320
  KC_KSET 322, 363
  return codes 319
kc_msg_dest_all_par 640
kc_mux_str 505
KC_NAME 279, 284
KC_NAME_NEXT 279, 284
kc_online_import_str 400
kc_osi_association_str 509
kc_osi_con_str 511
kc_osi_lpap_str 517
kc_pagepool_str 641
KC_PROGRAM 192
  return codes 235, 264
KC_PROGRAM_STMT 251
kc_program_str 205, 523
KC_PTC 287
KC_PTC_TA 165, 401
KC_PTERM 192
  return codes 236, 265
kc_pterm_str 207, 529
kc_queue_par_str 643
kc_queue_str 543
KC_RC_NIL 163
KC_READ_NO_GSSBFILE 285
KC_READ_NO_USERFILE 286
KC_SEND_MESSAGE 404
  return codes 408
KC_SFUNC 287
kc_sfunct_str 544
KC_SHUTDOWN 409
  return codes 416
kc_shutdown_str 415
KC_SPOOLOUT 418
  return codes 421
KC_SUBNET 287
kc_subnet_str 546

KC_SYSLOG 422
return codes 428
kc_syslog_str 430
kc_system_par_str 648
KC_TAC 192
return codes 240
kc_tac_str 214, 547
kc_tacclass_str 559
kc_tasks_par_str 652
kc_timer_par_str 656
kc_tpool_str 562
kc_transfer_syntax_str 572
kc_triple_str 171
KC_UPDATE_IPADDR 433
return codes 438
KC_USER 192, 622
return codes 243, 266
KC_USER_DYN1 287
kc_user_dyn1_str 288, 573
KC_USER_DYN2 287
kc_user_dyn2_str 288, 573
KC_USER_FIX 287
kc_user_fix_str 288, 573
KC_USER_STMT 251
kc_user_str 222, 288, 573
KC_USLOG 440
return codes 442
kc_utmd_par_str 661
KC_VERSION_DATA 163
kcadminc.h 443
KDCADMI
calling repeatedly 116
operation codes 164
return codes 174
trace 369, 614
KDCADMI function call
C/C++ 161
COBOL 842
KDCAPLKS 473
KDCAPPL 666
output 679
performance monitoring enable/disable 678
KDCBNDL 680
KDCDADM 821
deleting asynchronous jobs 822
deleting messages 822
entering in the configuration 820
KDCDEF generation
CALLUTM 856
for dynamic configuration 69
recommendations for 107
KDCDEF statements
generating 101, 246
KDCDIAG 681
output 689
KDCFILE
base name 632
catalog ID 629
inverse KDCDEF 252
redundant 632
reserving table spaces 69
kdcfile_name 632
kdcfile_operation 632
KDCHELP 690
KDCINF 691
controlling output 701
large values in floating point presentation 692
output 703
kdcload 650
KDCLOG 742
KDCLPAP 743
output 749
KDCLSES 751
output 752
KDCLTAC 753
output 754
KDCLTERM 755
output 757
KDCMON
activating/deactivating (KDCDIAG) 684
activating/deactivating (program) 375
KDCMUX 758
output 761
KDCPADM
entering in the configuration 820

KDCPOOL 762
output 763
KDCPROG 764
output 767
KDCPTERM 769
output 773
KDCS message area
length 633
KDCSEND 404, 690, 774
output 774
KDCSHUT 775
KDCSIGN 629
KDCSLOG 779
output 781
KDCSWTCH 785
output 787
KDCTAC 789
output 791
KDCTCL 792
example 796
output 795
KDCUSER 797
output 798
Kerberos principal
maximum length 639
kerberos_dialog 502
kc_lterm_str (KC_CREATE_OBJECT) 204
kc_tpool_str 571
key code
for a key set 474
largest 632
of a key set 195
key set
data structure (properties) 473
defining properties (program) 195
deleting dynamically 93
dynamic modification 363
entering dynamically 195
information about (KDCINF) 695
information about (program) 287
KDCAPLKS 473
key codes contained 195
key codes included in 474
LTERM partner 493
key set (cont.)
LTERM pool 565
LU6.1 partner application 479
master 195, 473
modifying dynamically 98, 322
OSI TP partner application 519
user ID 578
key sets
entering dynamically 79
keys 474
KC_MODIFY_OBJECT 322
keyvalue 632
keyword operands
administration commands 665
keywords
passing (KDCADMI) 177
ks_deleted
kc_kset_str 474
ks_name 473
kset
kc_lpap_str 479
kc_lterm_str 493
kc_lterm_str (KC_CREATE_OBJECT) 200
kc_osi_lpap_str 519
kc_tpool_str 565
kc_user_str 578
kc_user_str (KC_CREATE_OBJECT) 222
kc_user_str (KC_MODIFY_OBJECT) 357
KSET with inverse KDCDEF 102
kset_free
kc_dyn_par_str 622
kset_total
kc_dyn_par_str 622
KTA trace in test mode 685

L

language code
application 632
LTERM partner 493
LTERM pool 565
message module 504
user ID 581

language environment
defining, LTERM partner 200
LTERM partner 493
LTERM pool 565
message module 504
standard for the application 632
last_switch_ok 431
lc_name 487
letters
kc_con_str 464
kc_mux_str 507
kc_pterm_str 538
lib
kc_load_module_str 476
kc_message_module_str 504
kc_program_str 525
library
load module 525
user message module 504
lifetime
statistical data 731
Linux distribution 15
LIST 701
listener ID
BCAMAPPL name 455
OSI TP access point 451
listener port number
BCAMAPPL name 455
LU6.1 partner application 463
OSI TP access point 451
listener_id
kc_access_point_str 451
kc_bcamappl_str 455
listener_port
kc_bcamappl_str 455
kc_con_str 463
kc_osi_con_str 515
kc_pterm_str 537
kc_pterm_str (KC_CREATE_OBJECT) 210
kc-access_point_str 451
little endian 142
lm_name 475
lnetname 479
load
current 727
maximum 727
load control, via programming interface 149
load mode
load module/shared object 476, 524
load module
data structure (properties) 475
information about (KDCINF) 695
information about (program) 287
load mode 476, 524
marked for replacement 477
name 475
program unit/VORGANG exit 524
properties (data structure) 475
public slice 525
replaceable 477
replacing with KDCPROG 764
replacing with program 179, 323
version 475
load_mode 476, 524
load_module 524
KC_CREATE_OBJECT 206
load_module_gen 650
load-dependent control 148
local access point
for OSI TP connection 512
properties (data structure) 447
local secondary storage area
maximum number 633
local session name 484
local_access_point 512
locale
application 632
LTERM partner 493
LTERM pool 565
message module 504
user ID 226, 581
locale_ccsname
kc_lterm_str 493
kc_max_par_str 632
kc_tpool_str 565
kc_user_str 581
kc_user_str (KC_CREATE_OBJECT) 226

locale_lang_id
kc_lterm_str 493
kc_max_par_str 632
kc_message_module_str 504
kc_tpool_str 565
kc_user_str 581
kc_user_str (KC_CREATE_OBJECT) 226
locale_terr_id
kc_lterm_str 493
kc_max_par_str 632
kc_message-module_str 504
kc_tpool_str 565
kc_user_str 581
kc_user_str (KC_CREATE_OBJECT) 226
locin 470
lock
releasing (cluster user file) 306
lock code
largest 632
LTAC 199, 490
LTERM partner, defining 200
LTERM pool 494, 565
transaction code 548
lock_code
kc_ltac_str 490
kc_lterm_str 494
kc_lterm_str (KC_CREATE_OBJECT) 200
kc_tac_str 548
kc_tac_str (KC_CREATE_OBJECT) 214
kc_tpool_str 565
lock/key code concept 200, 493
log_rec_pages 642
logackwait_sec 659
KC_MODIFY_OBJECT 383
logfile_writes 606
kc_curr_par_str
(KC_MODIFY_OBJECT) 366
logical connection 518
re-establishing 382, 657
low 470
lowest_open_gen 432
lp_name 479
lpap
kc_con_str 463
kc_lses_str 484
kc_ltac_str 487
LPAP partner
administering (KDCLPAP) 744
administering (program) 325
information about (KDCINF) 698
information about (program) 287
information on (program) 293
LU6.1 partner application 463
LU6.1 session 484
partner application 193
properties (data structure) 478
remote service (LTAC) 197, 487
LPUT records
buffer size 633
length of user data 633
lput_pages 642
lputbuf 633
lputlth 633
ls_deleted
kc_lses_str 486
ls_name 484
lses_total
kc_dyn_par_str 623
LSSB
maximum number 633
lssb_pages 641
lssbs 633
lt_group 501
lt_name 493
KC_CREATE_OBJECT 200
LTAC
deleting dynamically 94
disabling/enabling (KDCLTAC) 753
disabling/enabling (program) 331
entering dynamically 81, 197
information about (KDCINF) 699
information about (program) 287
modifying properties (KDCLTAC) 753
modifying properties (program) 330
name 487
number of jobs 490

LTAC (cont.)
wait time for response 198, 490
with inverse KDCDEF 102
ltac_freel
kc_dyn_par_str 623
ltac_total
kc_dyn_par_str 623
ltac_type 490
ltacunit 491
LTERM
privileged for WinAdmin/Webadmin 124
lterm 534, 563
KC_CREATE_OBJECT 208
KC_MODIFY_OBJECT 344
LTERM bundle
exchange master LTERM 334, 680
LTERM group
assign primary LTERM 334
LTERM partner
adding dynamically 74, 188, 200
assignment to client/printer (KDCSWTCH) 786
assignment to client/printer (program) 343
define properties 200
deleting dynamically 87, 256
disable/enable (KDCLTERM) 756
disable/enable (program) 332
disabling 65
establishing/shutting down a connection (KDCLTERM) 755
establishing/shutting down a connection (program) 332
information about (KDCINF) 695
information about (program) 287, 294, 297
LTERM pool 500
maximum number 620
modifying dynamically 96
modifying properties (KDCLTERM) 755
modifying properties (program) 332
number that can be added dynamically 620
printer pool 500
properties (data structure) 492
statistical information 501
LTERM pool
access privileges 565
active clients 568
administering (KDCPOOL) 762
administering (program) 355
changing number of clients (KDCPOOL) 762
changing number of clients (program) 355
information about (KDCINF) 696
information about (program) 287
key set 565
maximum number of clients 564
multiple connections, client 564
prefix 563
properties (data structure) 562
start format 566
LTERM prefix
LTERM pool 563
LTERM statements
generating 101
lterm_curr 584
lterm_free 620
lterm_total 620
LU6.1 connection
adding dynamically 193
current status 464
deleting dynamically 93
duration of 464
entering dynamically 80
loss of 465
number of messages 464
pacing value 481
properties (data structure) 462
QUIET status 482, 485
LU6.1 partner application
logical properties (data structures) 478
LU6.1 session
defining properties (program) 196
deleting dynamically 93
entering dynamically 80, 196
idle time, timer 481
modifying dynamically 99
properties (data structure) 484

M
magnetic stripe card, user ID 578
main code of return code 174
map
  kc_lpap_str 481
  kc_osi_con_str 515
  kc_pterm_str 535, 568
  kc_pterm_str (KC_CREATE_OBJECT) 209
mapped_name 546
master 473, 483, 498, 522
master key set 195, 473
master LTERM
  exchange 334, 680
MAX DESTADM 140
MAX parameters (data structure) 624
max_asyn_ta_per_100sec 604
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 365
max_cpgpool_size 362, 592
max_dial_step_per_100sec 604
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 365
max_dial_ta_per_100sec 604
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 365
max_load
  kc_curr_par_str 607
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 366
max_number 564
max_open_asyn_conv 639
max_pool_size 604
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 365
max_wait_resources 366, 607
max_wait_system_resources 367, 607
MAX-CONN-USERS 671
MAXASYN 670
maxcon 507, 568
maximum values for application
  change (KDCAPPL) 666
  data structure 624
  modifying (KDCAPPL) 666
  modifying (program) 377
maximum_jr 607
  kc_curr_par_str
    (KC_MODIFY_OBJECT) 366
maxjr 661
maxses 506
md_name
  kc_msg_dest_par_str 640
md_type
  kc_msg_dest_par_str 640
message
  CALLUTM 878
  maximum length 638
  number in message queue (OSI-LPAP) 521
  passing to BCAM 628
  send to user (KDCSEND) 774
  send to user (program) 404
message area, length 633
message destination
  MSGTAC 148
  properties 640
message dump 612
  creating (KDCDIAG) 682
message interface
  for central administration 144
message module
  properties (data structure) 503
message queue
  administering (DADM) 805
  control value (LTERM partner) 202, 496
  control value (LTERM pool) 567
  control value (LU6.1 partner) 480
  control value (OSI-LPAP partner) 521
  control value (TAC) 551
  deleting asynchronous jobs (DADM) 810
  deleting asynchronous jobs
    (KDCDADM) 822
  deleting messages (DADM) 810
  deleting messages (KDCDADM) 822
  displaying information(KDCDADM) 824
  identification 807
  information about (DADM) 808
  information about (KDCDADM) 824
  number of messages (LTERM partner) 501
  number of messages (LU6.1 partner) 482

message queue (cont.)
number of messages (OSI-LPAP) 521
number of messages (TAC) 554
reducing the size of 59
threshold value 496
messages
administering (KDCDADM) 821
dead letter queue (DADM) 811
deleting (DADM) 810
deleting (KDCDADM) 822
identifying 807
information about (KDCDADM) 824
prioritizing (DADM) 809
prioritizing (KDCDADM) 829
metasyntax 29
mg_name 309
mg_node 309
Micro Focus COBOL 236
Microfocus COBOL 236
mm_name 504
mod_free_dial_tasks 381, 655
mod_max_asyntasks 654
KC_MODIFY_OBJECT 380
mod_max_tasks 653
KC_MODIFY_OBJECT 380
mod_max_tasks_in_pgwt 654
KC_MODIFY_OBJECT 381
modify
application parameters (KDCAPPL) 666
application parameters (program) 312, 317
cache paging (KDCAPPL) 668
cache paging (program) 377
client 96
configuration 67
key set 98, 322, 363
LU6.1 session 99
maximum number of sign-on attempts 379
number of clients in LTERM pool 355, 762
object properties (program) 311
objects 95
output destination, asynchronous command 140
password for user (KDCUSER) 797
printer 96
modify (cont.)
printer assignment (KDCSWTCH) 785
printer assignment (program) 343
TAC queue 97, 347
transaction code 97
user ID 98
MODIFY-CONFIGURATION 870
monitoring time
establishing a connection (OSI TP) 384, 659, 660
establishing a session (LU6.1) 384, 659, 660
mp_wait_sec 633
MSCF 889
MSGTAC message destination 148
msgtac_pages 642
multi-step service, administration program 117
multi-step transaction
setting timer (KDCAPPL) 678
setting timer (program) 383
timer 659
multiplex connection
connection status 507
disabling/enabling (KDCMUX) 761
disabling/enabling (program) 336
establishing/shutting down connection (KDCMUX) 759
establishing/shutting down connection (program) 337
information about (KDCINF) 694, 696
information about (program) 287, 295
maximum number of clients 506
maximum number of terminals (KDCMUX) 760
maximum number of terminals (program) 336
modifying properties (KDCMUX) 758
modifying properties (program) 336
number of active clients 507
number of exchanged messages 507
properties (data structure) 505
statistical information 507
mux 538
MUX, see multiplex connection
mx_name 506

N
name
client/printer 530
common memory pool 525
format 84
format of data passed in data structures 444
key set 473
load module/shared object 475
local application 454, 602, 627, 649
LPAP partner 479
LTAC 487
LTERM partner 493
LU6.1 session 196
message module 504
multiplex connection 506
OSI TP access point 447
OSI TP connection 512
partner application 193
program unit/VORGANG exit 523
transaction code 548
UTM user ID 578
name class 83
name list
output (KDCINF) 701
nb 633
length 633
nbr_ack_jobs
kc_tac_str 558
nbr_dputs 522
kc_lpap_str 483
kc_lterm_str 501
kc_tac_str 557
nbr_ta_commits 558
net_access 633
NETCOBOL 236
netprio
kc_lpap_str 479
kc_lterm_str 497
kc_lterm_str (KC_CREATE_OBJECT) 203
kc_mux_str 507
kc_tpool_str 567
network component
OSI TP address 515
network_selector 515
new password
user ID (KDCUSER) 797
user ID (program) 359
node application
information on (program) 287
reference name 461
node_indx 458
node_name
kc_cluster_str 461
kc_lses_str 486
node_reserved_cpgpool_pages 592
nolog 470
notational conventions 29
CALLUTM statements 861
data structure description 445
nr_cache_rqs 367, 608
nr_cache_searches 367, 608
nr_calls 561
nr_res_rqs 608
nr_res_rqs_for_max 367, 608
nr_sys_res_rqs 608
nr_sys_res_rqs_for_max 367, 609
nr_waits 561
nrconv 633
number of processes
asynchronous processing, setting 654
blocking calls, maximum 653
number_errors 554
number_errrors_ex 558
number_ret 170
number_tacs 585
numeric data in data structures 444
numeric values
converting 146
numerical data, passing (KDCADMI) 177

O
obj_number 170
obj_type
KC_GET_OBJECT 287
parameter area 166
object name
format/uniqueness 82
passing (KDCADMI) 171, 177

object properties
modifying 311
passing (KDCADMI) 173
object table, reserving spaces 69
object type 166, 316
objects
adding dynamically 73, 188
deleting dynamically 85, 256
dynamic creation/deletion 67
modifying dynamically 95
number of 170
oc_name 512
offline, inverse KDCDEF 104
ohdr 470
ol_name 518
ONCALL 476, 524
online import
application data 398
asynchronous service 398
source node (data structure) 400
start node (data structure) 172, 402
online, inverse KDCDEF 103
open_asyn_services 604
open_dial_services 604
openSM2
KDCAPPL 678
openSM2 data supply 636
activating/deactivating (KDCAPPL) 678
activating/deactivating (program) 379
enable/disable 678
operating system 649
operation codes KDCADMI 164
options
databases 466
edit profiles 468
OSI TP access point
for OSI TP connection 512
properties (data structure) 447
OSI TP association
idle time, timer setting 520
information about (KDCINF) 699
information about (program) 289
prefix 518
properties (data structure) 509
OSI TP connection
diagnosis (KDCDIAG) 687
diagnosis (program) 371
duration 510
information about (KDCINF) 699
information about (program) 287
number of parallel associations 519
properties (data structure) 511
status 515
status QUIET 521
OSI TP partner address 512
OSI TP partner application
properties (data structure) 517
OSI TP replacement connection
activating (KDCLPAP) 745
activating (program) 338
OSI trace 612
activating/deactivating (KDCDIAG) 687
activating/deactivating (program) 370
osi_con 521
osi_lpap
kc_osi_association_str 509
kc_osi_con_str 512
osi_scratch_area 633
osi_tp 651
osi_trace 612
KC_MODIFY_OBJECT 371
osi_trace_records 613
KC_MODIFY_OBJECT 371
OSI-LPAP partner
administering (KDCLPAP) 744
administering (program) 339
disable/release (KDCLPAP) 748
disable/release (program) 339
information about (KDCINF) 699
information about (program) 287
OSI TP association 509
OSI TP connection 512
osishmkey 634
other_pages 642
OTRACE 687

out_queue
kc_lpap_str 482
kc_lterm_str 501
kc_osi_lpap_str 521
kc_user_str 587
out_queue_ex
kc_lpap_str 483
kc_lterm_str 501
kc_osi_lpap_str 522
kc_user_str 590
output
asynchronous command 113
dialog command 110
KDCAPPL 679
KDCDIAG 689
KDCINF 703
POOL 718
KDCINF CON 704
KDCINF KSET 705
KDCINF LOAD-MODULE 706
KDCINF LPAP 708
KDCINF LSES 709
KDCINF LTAC 710
KDCINF LTERM 711
KDCINF MUX 713
KDCINF OSI-ASSOCIATIONS 715
KDCINF OSI-CON 716
KDCINF OSI-LPAP 717
KDCINF POOL 720
KDCINF PROG 721
KDCINF PTERM 723
KDCINF SHARED-OBJECT 725
KDCINF STATISTICS 726
KDCINF SYSPARM 733
KDCINF TAC 737
KDCINF TAC-PROG 738
KDCINF TACCLASS 739
KDCINF USER 740
KDCLPAP 749
KDCLSES 752
KDCLTAC 754
KDCLTERM 757
KDCMUX 761
output (cont.)
KDCPOOL 763
KDCPROG 767
KDCPTERM 773
KDCSEND 774
KDCSLOG 781
KDCSWTCH 787
KDCTAC 791
KDCTCL 795
KDCUSER 798
output debug information (program) 376
output destination
asynchronous command 140

P

paccnt 481
pacing value, LU6.1 connection 481
PADM
functions 800
program unit 820
sample programs 820
page pool 633
avoiding overflows 57
node application 62
number of files 634
properties 634
size of 634
utilization 365, 604
warning levels 634
pages_pwrite 606
kc_curr_par_str
(KC_MODIFY_OBJECT) 366
paging
cache 628
controlling cache (KDCAPPL) 668
controlling cache (program) 377
PAM read/write jobs
maximum number 635
parallel connections
clearing (KDCLPAP) 748
clearing (program) 342, 771
number (OSI TP) 519
OSI-LPAP partner (KDCLPAP) 746
OSI-LPAP partner (program) 340

parallel OSI TP associations
number 519
parameter area 162
conversion 146
data_lth 170
data_lth_ret 170
id_lth 170
number_ret 170
obj_number 170
obj_type 166
retcode 163
select_lth 170
subopcode 164
version 163
version_data 163
parameter types 168
parameter values
passing (KDCADMI) 173
parameters for distributed processing
data structure 661
partner application
information about (KDCINF) 699
information about (program) 294, 295
LU6.1, logical properties 478
partner type
client/printer 531
LTERM pool 564
passing application parameters 591
password 579
change with maximum term of validity 360
changing (KDCUSER) 797
changing (program) 358
complexity level 582
darkened entry 580
enter in a darkened field 225
kc_user_str (KC_CREATE_OBJECT) 224
kc_user_str (KC_MODIFY_OBJECT) 359
minimum length 581
validity, duration 582
password_dark 580
KC_CREATE_OBJECT 225
password_type 580
kc_user_str (KC_CREATE_OBJECT) 225
kc_user_str (KC_MODIFY_OBJECT) 359
PCMX 20
PDN 533
performance bottleneck, measures 51
performance check 48
performance monitoring
with openSM2 (KDCAPPL) 678
with openSM2 (program) 379
period of validity of modification
UTM cluster application 314
periodic write 366, 606
periodic_writes 606
kc_curr_par_str
(KC_MODIFY_OBJECT) 366
permit
kc_lpap_str 479
kc_osi_lpap_str 520
kc_user_str 583
kc_user_str (KC_CREATE_OBJECT) 227
pgpool_pages 634
pgpool_warnlevel1 634
pgpool_warnlevel2 634
pgpoolfs 634
pgwt 560, 650
PGWT TAC class 560
PGWT wait time 658
setting timer (KDCAPPL) 671
setting timer (program) 382
PGWTTIME 671
pgwttime_sec 658
KC_MODIFY_OBJECT 382
phys_msg_pages 642
pisizelth 634
plev 496
KC_CREATE_OBJECT 202
plu 481
polling function
central administration 133
POOL 476, 525
pool 500, 538
poolname
kc_load_module_str 477
kc_program_str 525

port number
BCAMAPPL name 455
client 537
LU6.1 partner application 463
OSI TP access point 451
OSI TP partner address 515
portable administration program 142
position operand
administration commands 665
pr_name 523
KC_CREATE_OBJECT 205
prefix
LTERM partner for a pool 563
OSI TP associations 518
prepare to commit 401
presentation selector
OSI TP access point 448
presentation_selector
kc_access_point_str 448
kc_osi_con_str 512
presentation_selector_code
kc_access_point_str 449
kc_osi_con_str 513
presentation_selector_lth
kc_access_point_str 449
kc_osi_con_str 513
presentation_selector_type
kc_access_point_str 449
kc_osi_con_str 513
previous version
load module/shared object 477
primary logical unit
LU6.1 session 481
princ_free
kc_dyn_par_str 623
princ_total
kc_dyn_par_str 623
principal
kc_user_str 589
principal_lth 639
print output
automatic mode 816
confirmation mode 816
confirming (KDCPADM) 836
print output (cont.)
confirming (PADM) 818
fault handling 819
hardware fault, action 819
repeating (KDCPADM) 836
repeating (PADM) 818
PRINTABLE-STRING 197, 489
printer
adding dynamically 74, 207
administering (KDCPADM) 830
administering (PADM) 800, 812
availability 66
changing an LTERM partner (program) 343
changing LTERM partner (KDCPADM) 838
changing LTERM partner (KDCSWTCH) 785
changing LTERM partner (PADM) 814
computer name 499
connection status 537
deleting dynamically 87, 259
disable/re-enable (KDCPADM) 837
disabling 65
disabling (PADM) 814
establishing a connection (KDCAPPL) 676
establishing a connection (program) 418
information about (KDCINF) 711
information about (KDCPADM) 831
information about (PADM) 813
information about (program) 296
maximum number in configuration 620
modifying dynamically 96
number that can be added dynamically 620
properties (data structure) 529
statistical information 538
printer control (KDCPADM) 830
printer control (PADM) 812, 815
printer control LTERM 495, 802
assigning (KC_CREATE_OBJECT) 201
printer control, printer ID 535
printer ID 535, 803
printer pool
generating 65
LTERM partner 500
printer queue control value 496

printer status
changing (KDCPADM) 837
changing (PADM) 814
prio 561
priority control 561
privilege keys, semaphores 635
privileged LTERM 124
privileged_lterm 639
processes
define current number (KDCAPPL) 677
maximum number 637, 653
number currently set 380
processes, defining number
for asynchronous processing (KDCAPPL) 670
for asynchronous processing (program) 380
for blocking calls (KDCAPPL) 677
for blocking calls (program) 381
in application (KDCAPPL) 677
in application (program) 380
TAC classes (KDCTCL) 792
TAC classes (program) 351
processes, number of
asynchronous processing 603
asynchronous processing, maximum 627, 653
changing for the application 51
current, defining 677
current, determining 603, 653, 655
data structure 652
maximum defined 603
maximum, for asynchronous processing 655
modifying for asynchronous processing 52
modifying TASKS-IN-PGWT 52
TAC class 560
processing time
program unit (KDCAPPL) 675
program unit (program) 383
processor of partner application 193
PROG 696
prog_change_running 650
PROGRAM 672
program 548
KC_CREATE_OBJECT 214
program interface
summary 37
program library
load module 476
program replacement
application (KDCAPPL) 672
application (program) 181
load module/shared object (KDCPROG) 764
load module/shared object/DLL 323
requirements 180, 672
several applications simultaneously 134
PROGRAM statements
generating 101
program statements
CALLUTM 860
program unit
adding dynamically 70, 78, 188, 205
average runtime 348, 554
deleting dynamically 89, 259
information about (program) 296
information on (KDCINF) 721
maximum number 621
number that can be added dynamically 621
processing time (KDCAPPL) 675
processing time (program) 383
properties (data structure) 523
program_fgg 607
program_fgg_new 185
program_fgg_old 185
program_free 621
program_total 621
pronam
kc_con_str 463
kc_lses_str 485
kc_lses_str (KC_MODIFY_OBJECT) 328
kc_lterm_str 499
kc_mux_str 506
kc_pterm_str 530
kc_pterm_str (KC_CREATE_OBJECT) 207
kc_tpool_str 563
properties
UTM cluster application 593

properties, modifying
client/printer (KDCPTERM) 769
client/printer (program) 343
LTAC (KDCLTAC) 753
LTAC (program) 330
LTERM (KDCLTERM) 755
LTERM (program) 332
multiplex connection (KDCMUX) 758
multiplex connection (program) 336
TAC (KDCTAC) 789
TAC (program) 347
protect_pw_compl 582
KC_CREATE_OBJECT 227
protect_pw_lth 581
KC_CREATE_OBJECT 226
protect_pw_min_time 587
protect_pw_time 582
KC_CREATE_OBJECT 227
protect_pw_time_left 360, 585
protocol
kc_pterm_str 536
kc_pterm_str (KC_CREATE_OBJECT) 210
kc_tpool_str 568
pt_name 530
KC_CREATE_OBJECT 207
PTC
determining transaction 590
rolling back transaction 401
ptc
kc_lses_str 486
kc_user_str 590
PTC wait time
setting timer (KDCAPPL) 674
setting timer (program) 384
PTCTIME
KDCAPPL 674
program 384
ptctime_sec 660
KC_MODIFY_OBJECT 384
PTERM
automatic connection setup 344, 772
change LTERM partner (KDCSWTCH) 785
change LTERM partner (program) 343
disabling with a program 345
PTERM (cont.)
disabling with KDCPTERM 771
establishing/shutting down a connection (program) 345
establishing/shutting down connection (KDCPTERM) 771
information about (KDCINF) 696
information about (program) 287
releasing session (KDCPTERM) 772
pterm 498
PTERM properties
modifying (KDCPTERM) 769
modifying (program) 343
PTERM statements
generating 101
pterm_free 620
pterm_total 620
ptype
KC_CREATE_OBJECT 208
kc_pterm_str 531
kc_tpool_str 564
supported partner types 531
ptype_class
kc_pterm_str (KC_CREATE_OBJECT) 208
public key
read 271
public slice
load module 476, 525
pw_history 645

Q

q_mode
kc_queue_str 543, 643
kc_tac_str 557
kc_user_str 588
q_read_acl
kc_tac_str 557, 587
q_write_acl
kc_tac_str 557
kc_user_str 588
qamsg 496
KC_CREATE_OBJECT 202

qlev
kc_lpap_str 480
kc_lterm_str 496
kc_lterm_str (KC_CREATE_OBJECT) 202
kc_osi_lpap_str 521
kc_queue_str 543, 643
kc_tac_str 551
kc_tac_str (KC_CREATE_OBJECT) 217
kc_tpool_str 567
kc_user_str 587
qp_number
kc_queue_par_str 643
qtime1, kc_timer_par_str 660
qtime2, kc_timer_par_str 660
qu_name
kc_queue_str 543
queue
properties (data structure) 543
service-controlled 805
temporary 805
UTM-controlled 805
queue_freel
kc_dyn_par_str 623
queue_length
kc_queue_str 543
queue_total
kc_dyn_par_str 623
QUIET status
LU6.1 connection 482, 485
OSI TP connection 521
quiet_connect
kc_lpap_str 482
kc_lpap_str (KC_MODIFY_OBJECT) 327
kc_lses_str 485
kc_lses_str (KC_MODIFY_OBJECT) 329
kc_osi_lpap_str 521

R

re-establish
connection (KDCAPPL) 669
connection (program) 382, 657
Readme files 22
real time
maximum, for program unit run 552
real_time_sec 552
KC_CREATE_OBJECT 217
recbuf_lth 634
recbuf_pages 634
recbuffs 635
recipient
results of asynchronous commands 114
recipient TPSU title 488
record type
OSI trace 613
Red Hat 15
regeneration, recommendations 107
relevant_bits 546
remote service
local TAC name 487
remote session name 484
replace
application (program) 181
application program (KDCAPPL) 672
load module/shared object (KDCPROG) 764
load module/shared object/DLL 323
replaceable
load module/shared object 477
replaceable programs
information about (KDCINF) 696
information about (program) 296
replacement connection
activating (KDCLPAP) 745
activating (program) 338
replywait_sec 490
reqnr 635
request_calls 510
RESERVE 69
reserve node application, assigning values 316
reserved names 82
reserving table spaces 69
reset_msg_pages 642
resource lock
setting timer (KDCAPPL) 675
setting timer (program) 382
wait time for 658
restart 497, 582
kc_lterm_str (KC_CREATE_OBJECT) 203
kc_user_str (KC_CREATE_OBJECT) 227

restart area
buffer size 634
number of files 635
size per process 634
restartable
central administration with client 128
restricted 645
result output
asynchronous command 113
dialog command 110
reswait_pr_sec 658
KC_MODIFY_OBJECT 383
reswait_ta_sec 658
KC_MODIFY_OBJECT 382
RESWAIT-PR 675
RESWAIT-TA 675
retcode 163
return code
data structure 174
KC_CHANGE_APPLICATION 184, 185
KC_CON (KC_CREATE_OBJECT) 231
KC_CON (KC_DELETE_OBJECT) 263
KC_CREATE_OBJECT 230
KC_CREATE_STATEMENTS 192, 254
KC_DELETE_OBJECT 262
KC_ENCRYPT 275
KC_GET_OBJECT 298
KC_KSET (KC_CREATE_OBJECT) 232
KC_KSET (KC_DELETE_OBJECT) 263
KC_LSES (KC_CREATE_OBJECT) 232
KC_LSES (KC_DELETE_OBJECT) 263
KC_LTAC (KC_CREATE_OBJECT) 232
KC_LTERM (KC_DELETE_OBJECT) 264
KC_MODIFY_OBJECT 319
KC_PROGRAM (KC_CREATE_OBJECT) 235
KC_PROGRAM (KC_DELETE_OBJECT) 264
KC_PTERM (KC_CREATE_OBJECT) 236
KC_PTERM (KC_DELETE_OBJECT) 265
KC_SEND_MESSAGE 408
KC_SHUTDOWN 416
KC_SPOOLOUT 421
KC_SYSLOG 428
return code (cont.)
KC_TAC (KC_CREATE_OBJECT) 240
KC_UPDATE_IPADDR 438
KC_USER (KC_CREATE_OBJECT) 243
KC_USER (KC_DELETE_OBJECT) 266
KC_USLOG 442
KDCADMI 174
rnetname 480
roll back
distributed transaction 662
transaction in PTC state 401
roll back transaction 401
RSA 268
RSA key pair
delete (activated) 271
enable 271
generate 271
read public key 271
rses 484
RSET 662
in the administration program 117
rset 662
RSO printer
adding dynamically 207
rtac 488
RTAC name 488
code type 197, 488
rtac_lth 488
rtryi 508
rtryo 508
run priority, transaction code 552
runprio 552
KC_CREATE_OBJECT 218

S

saml 471
sample programs 41
sat 635
SAT administration interface 119
SAT administration privileges
OSI TP application 520
remote application (LU6.1) 227, 479
transaction code 553
user ID 583

SAT logging
administering 31
application 635
TAC-specific 553
user-specific 583
satadm 553
KC_CREATE_OBJECT 218
satsel
kc_tac_str 553
kc_tac_str (KC_CREATE_OBJECT) 219
kc_user_str 583
kc_user_str (KC_CREATE_OBJECT) 228
scope 416
screen output
properties, edit profile 468
SDF command interface
central administration 128
seccounter
kc_lterm_str 501
kc_user_str 585
security violations
number of (LTERM partner) 501
select_lth 170
SELECT-SERVICE 871
selection area 172
length 170
semaphore 635
global, privilege key 635
semkey 636
service exit 551
service restart 203
LTERM partner 497
user ID 582
service stacking
maximum depth 633
service TAC 551
services
disabling 56
number currently open 604
number of abnormally terminated 366
number terminated abnormally 605
remote, local TAC name 487
session
administering (KDCLSES) 751
administering (program) 328
information about (program) 287
local, information about (KDCINF) 698
LU6.1, information about (program) 294
maximum number, multiplex connection 506
properties (data structure) 484
reference name of node application 486
timer for assignment 489
timer for idle state 481
timer for idle state (KDCLPAP) 748
timer for idle state (program) 326
timer for occupation 198
session name 196, 484
session selector
OSI TP access point 448
OSI TP partner address 512
session setup
global monitoring time (program) 384
setting the timer (KDCAPPL) 669
session_selector
kc_access_point_str 448
kc_osi_con_str 512
session_selector_code
kc_access_point_str 450
kc_osi_con_str 514
session_selector_lth
kc_access_point_str 450
kc_osi_con_str 514
session_selector_type
kc_access_point_str 450
kc_osi_con_str 514
several administration calls 116
shared memory segment
access key 631
privilege keys 628
shared object
information about (KDCINF) 696
load mode 524
name 475
properties (data structure) 475
replaceable 477
replacing with a program 323

shared object (cont.)
replacing with KDCPROG 764
version 475
shortage (BCAM bottleneck) 508
SHOW-CONFIGURATION 875
shut down a session (program) 329
sign_time_date 586
sign-on
restrictions 636
sign-on attempts
modifying the maximum number 379
sign-on procedure (SIGNON)
information on (program) 288
SIGNON, information on (program) 288
signon_fail 636
KC_MODIFY_OBJECT 379
signon_restr 636
signon_tac
kc_bcamappl_str 456
signon_value 636
silent_alarm 646
simultaneous administration
several applications 133, 138
size control, activating
SYSLOG file (KDCSLOG) 779
SYSLOG file (program) 431
size_control_engaged
(KC_SYSLOG) 431
size_control_kbyte
(KC_SYSLOG) 432
size_control_suspended
(KC_SYSLOG) 431
size_control_utmpages
(KC_SYSLOG) 432
sm2 636
KC_MODIFY_OBJECT 379
socket application 533, 534
Solaris 15
sort order 142
SPAB
clear character 629
length 637
spab 637
specin 471
SPOOLOUT 676
standalone UTM application 13
standard 504
standard language environment 632
standard primary working area
clear character 629
maximum length 637
standard UTM message module 504
start format 495, 566
LTERM partner
(KC_CREATE_OBJECT) 202
LTERM partner
(KC_MODIFY_OBJECT) 333
LTERM partner/terminal 495
LTERM pool 566
user ID 580
user ID (program) 361
start_date_... 602
start_gen 432
start_time_... 602
start, inverse KDCDEF 103
STARTUP 476, 524
state
kc_con_str 464
kc_lpap_str (KC_MODIFY_OBJECT) 325
kc_ltac_str 489
kc_ltac_str (KC_MODIFY_OBJECT) 331
kc_lterm_str 494
kc_lterm_str (KC_CREATE_OBJECT) 201
kc_lterm_str (KC_MODIFY_OBJECT) 332
kc_mux_str 507
kc_mux_str (KC_MODIFY_OBJECT) 336
kc_osi_lpap_str 339, 520
kc_pterm_str 534
kc_pterm_str (KC_CREATE_OBJECT) 209
kc_pterm_str (KC_MODIFY_OBJECT) 345
kc_tac_str 549
kc_tac_str (KC_CREATE_OBJECT) 215
kc_tac_str (KC_MODIFY_OBJECT) 347
kc_tpool_str 566
kc_tpool_str (KC_MODIFY_OBJECT) 355
kc_user_str 578
kc_user_str (KC_CREATE_OBJECT) 222
kc_user_str (KC_MODIFY_OBJECT) 357

state_number 566
KC_MODIFY_OBJECT 355
STATIC 476, 525
statistical information
client/printer 538
distributed processing 606
KDCINF 697
load on the application 600
LTERM partner 501
multiplex connection 507
OSI TP association 510
transaction code 554
statistics information
user ID 584, 585
status
client connection 537
client/printer 534
connection to client/printer 500
connection to message router 507
LPAP partner 464, 480
LTAC 489
LTERM partner 494
LTERM partner, defining (program) 201
LTERM partners in a pool 566
LU6.1 connection 482
multiplex connection 507
OSI TP association 510
OSI TP connection 515
OSI-LPAP partner 520
user ID 578
STATUS, KDCLPAP 748
structure
administration program 115
stxit_log
kc_diag_and_account_par_str 614
KC_MODIFY_OBJECT 376
STXIT-LOG
enabling (KDCDIAG) 688
subcode of return code 174
subopcode
parameter area 164
SUSE 15
SUSRMAX 845
switch
SYSLOG file (KDCSLOG) 780
SYSLOG file (program) 423
user log file (KDCLOG) 742
user log file (program) 440
switch over log files (KDCAPPL) 676
switch over log files (program) 375
synchronous commands
for administration 140
syntax, querying, administration commands 690
SYSLOG
administering (KDCSLOG) 779
administering (program) 422
information about (KDCINF) 697
information about (program) 422
syslog_size 637
SYSPARM 697
SYSPROT 676
sysprot_switch
kc_diag_and_account_par_str 613
KC_MODIFY_OBJECT 375
system environment
CALLUTM 876
system parameters
data structure 648
information about (KDCINF) 697
information about (program) 288
system_type 649

T

t_prot
kc_access_point_str 451
kc_bcamappl_str 454
kc_con_str 463
kc_osi_con_str 516
kc_pterm_str 537
T61-STRING 197, 489
table spaces
reserving 69
TAC class 792
average wait time 353, 560
for blocking calls 560
information about (KDCINF) 697
information about (program) 287

TAC class (cont.)
modifying properties (KDCTCL) 792
modifying properties (program) 351
number of jobs pending 560
number of processes 560
of a transaction code 550
properties (data structure) 559
TAC for services, entering dynamically 197
TAC properties
changing (KDCTAC) 789
changing (program) 347
TAC queue 805
adding dynamically 214
data access control 221
modifying dynamically 97, 347
online import 398
TAC statements, generating 101
tac_elap_msec 554
tac_free 621
tac_total 621
tac_type
kc_tac_str 551
kc_tac_str (KC_CREATE_OBJECT) 217
TAC-PROG 697
TAC-specific SAT logging 553
tacclass 550, 559
KC_CREATE_OBJECT 216
tacclass_pages 642
tacclasses 649
taccpu_micro_sec 558
taccpu_msec 555
tacunit 553
KC_CREATE_OBJECT 219
task, see process
TASKS 677, 793
tasks
kc_max_par_str 637
kc_tacclass_str 352, 560
kc_tasks_par_str 653
tasks_free 560
KC_MODIFY_OBJECT 352
tasks_in_pgwt
kc_max_par_str 637
kc_tasks_par_str 653
tasks_waiting_in_pgwt 604
TASKS-IN-PGWT 677
TASKSFREE 794
tc_name 548
KC_CREATE_OBJECT 214
TCB entries (TAC) 553
tcbentry 553
term_input_msgs 603
kc_curr_par_str
(KC_MODIFY_OBJECT) 365
term_output_msgs 603
kc_curr_par_str
(KC_MODIFY_OBJECT) 365
terminal, adding dynamically 74, 207
terminal mnemonic
client/printer 536
LTERM pool 567
LU6.1 partner application 463
OSI TP partner application 521
terminal type
client/printer 531
LTERM pool 564
terminate
UTM application (KDCSHUT) 776
UTM application (program) 409
termn
kc_con_str 463
kc_osi_lpap_str 521
kc_pterm_str 536
kc_pterm_str (KC_CREATE_OBJECT) 209
kc_tpool_str 567
TERMWAIT 678
termwait_end_ta_sec 659
termwait_in_ta_sec 659
KC_MODIFY_OBJECT 383
territorial code
application 632
client/printer 493
LTERM partner 493
LTERM pool 565
message module 504
user ID 581

test mode 612, 681
activating/deactivating (KDCDIAG) 685
activating/deactivating (program) 373
TESTMODE 638, 685
testmode 612
KC_MODIFY_OBJECT 373
time 603
time_min 415
time-driven asynchronous jobs
execution time 630
number currently waiting 605
timer 656
assign an LU6.1 session 489
confirmation, asynchronous message (program) 384
entering dialog partner 356
entry from dialog partner 346
entry from dialog partner (program) 211
idle state, OSI TP association 520
idle state, session 481
input from dialog partner 659
multi-step transaction 659
occupation of LU6.1 session 198
PEND KP call 659
PEND KP calls (KDCAPPL) 678
PEND KP calls (program) 383
PGWT wait time (KDCAPPL) 671
PGWT wait time (program) 382
print confirmation 659
print confirmation (program) 383
re-establishing a connection (KDCAPPL) 669
re-establishing a connection (program) 382
resources locked 658
resources locked (KDCAPPL) 675
resources locked (program) 383
response from dialog partner 211, 539, 569
session setup 659
session/association assignment 489
session/association occupation 198
shutdown (KDCSHUT) 777
shutdown (program) 414
transport confirmation 659
transport confirmation (program) 383
timer (cont.)
wait time for blocking calls 658
wait time for response 198, 490
waiting in PTC (KDCAPPL) 674
waiting in PTC (program) 384
waiting in PTC state 660
timer settings 198
data structure 656
defining (KDCAPPL) 666
defining (program) 382
information about (KDCINF) 697
information about (program) 288
modifying 54
tls_pages 641
total_pages 641
TPOOL
administering (KDCPOOL) 762
administering (program) 355
information about (KDCINF) 718, 720
information about (program) 287
TRACE area, number of entries 638
trace, activating/deactivating
BCAM (KDCDIAG) 685
BCAM (program) 327, 369
OSI TP (KDCDIAG) 687
OSI TP (program) 370
tracerec 638
tracing in test mode 373, 685
transaction
determining in PTC state 590
distributed, rolling back 662
transaction code
access protection 548
adding dynamically 71, 78, 188, 214
deleting dynamically 89, 259
disable 549
disabling (KDCTAC) 789
disabling (program) 347
enabling (KDCTAC) 790
enabling (program) 347
in DESTADM 140
information about (KDCINF) 697
information about (program) 287, 297
lock code 548

transaction code (cont.)
maximum number 621
modifying dynamically 97
modifying properties (KDCTAC) 789
modifying properties (program) 347
name 548
number that can be added dynamically 621
properties (data structure) 547
remote service 487
statistical information 554
transaction management
central administration 133
during dynamic creation 73
during dynamic deletion 86
transfer syntax 572
transport confirmation 659
transport connection
defining 93
for LU6.1 session 485
OSI-LPAP partner 521
to LU6.1 partner, properties 462
transport priority
connections (LTERM pool) 567
defining (LTERM partner) 203
on client connection 497
on LU6.1 connection 479
on MUX connection 507
transport protocol 454, 568
on connection to client 536
transport selector
OSI TP access point 450
OSI TP partner address 514
transport_selector 450, 514
trmsglth 638
trusted client 212, 540, 570
TS application 533
adding dynamically 76, 207
creating, example 77
deleting dynamically 87
enter dynamically 76
properties (data structure) 529
tsel_format
kc_access_point_str 452
kc_bcamappl_str 456
tsel_format (cont.)
kc_con_str 464
kc_osi_con_str 516
kc_pterm_str 537
TX, trace 371
TX trace, enabling/disabling 371
TX trace function 615
tx_trace, KC_MODIFY_OBJECT 371
type
client/printer 531
transaction code 551

U

uls_pages 642
unproc_atacs 605
unproc_prints 605
update IP addresses 433
upic 646
UPIC client
add to the configuration dynamically 76
creating, example 77
dynamic deletion 87
UPIC program
adding dynamically 207
central administration 129
upicfile 130, 131
uppercase letters 142
us_name 578
kc_user_str (KC_CREATE_OBJECT) 222
usage_type
kc_lterm_str 494
kc_lterm_str (KC_CREATE_OBJECT) 201
kc_pterm_str 536
kc_pterm_str (KC_CREATE_OBJECT) 210
used 490, 554
user ID 797
access privileges 578
active (LTERM partner) 499
adding dynamically 72, 79, 188, 222
automatic KDCSIGN (program) 201
BCAM trace 361
changing password (KDCUSER) 797
changing password (program) 358
create for client programs 76

user ID (cont.)
creating with ID card 72
deleting dynamically 91, 259
deleting password (KDCUSER) 797
deleting password (program) 358
disabling (KDCUSER) 798
disabling (program) 357
for automatic KDCSIGN 75
for automatic KDCSIGN (LTERM) 494
ID card 578
information about (KDCINF) 740
information about (program) 297
issuing password (KDCUSER) 797
modifying dynamically 98
new start format 361
number that can be added dynamically 622
password 579
properties (data structure) 573
statistics information 584, 585
with ID card, maximum number 622
with ID card, number that can be added dynamically 622
user log file 440, 638
current file generation 607
double 638
name 632
switching (KDCLOG) 742
switching (program) 440
user message module
properties (data structure) 503
user properties
modifying (KDCUSER) 797
modifying (program) 357
USER queue 805
user service protocol
on connection to client/printer 536
on LTERM pool connections 568
USER statements
generating 101
user_curr 499
user_free 622
user_gen 494
KC_CREATE_OBJECT 201
user_kset 570
user_message 416
user_total 622
user_type 584
user-specific SAT logging 583
user-specific start format 580
users 485
active, current number 604
defining maximum number 55
defining maximum number (KDCAPPL) 671
defining maximum number (program) 378
information about (KDCINF) 698
information about (program) 287
maximum number 630
USLOG
base name 632
current file generation 607
double 638
uslog 638
uslog_fgg 607
usp_hdr
kc_pterm_str 541
kc_tpool_str 571
utilization page pool 604
UTM accounting phase
activating (KDCAPPL) 668
activating (program) 375
UTM accounting, see accounting
UTM administration
in dialog (commands) 110
UTM administration using message queuing 112
UTM application
terminating (KDCSHUT) 776
terminating (program) 413
UTM client
central administration 128
generating for administration 130
generating for administration (cluster) 131
UTM cluster application 13
adminstration 122
cluster administration journal 886
global properties 593
UTM dump
create with program 186
generating, KDCDIAG 682

UTM event monitor 611
activating/deactivating (KDCDIAG) 684
activating/deactivating (program) 375
UTM generation variant 602, 649
UTM information functions 46
UTM page
size 627
UTM system processes
current number 609, 655
maximum number 655
UTM user ID, see user ID
UTM version 602, 649
utm_version 602, 649
UTM-F application 602, 649
UTM-S application 602, 649
utmd 650

V

value
kc_insert_str 617
value_type
kc_insert_str 618
version 163, 475
KC_MODIFY_OBJECT 323
load module/shared object 475
of data structures 163
version_data 163
version_prev 477
vgmsize 638
VORGANG exit
dynamic creation 70, 78, 205
dynamic deletion 89
information about (program) 296
properties (data structure) 523
transaction code 551
VTAM name, LU6.1 partner 480

W

wait time
blocked resources 658
for response 198
for response, timer setting 490
job-receiver confirmation 660
PGWT 658
wait time (cont.)
session occupancy (KDCLTAC) 753
session occupancy (program) 330
session/association assignment 489
session/association occupation 198
to access a session (program) 330, 331
wait_dputs 605
wait_go 507
wait_resources 605
wait_system_resources 607
waiting_msgs 560
warning level
cluster page pool 62
page pool settings 634
WebAdmin description 123
weighting
LTAC for UTM Accounting 491, 553
LTAC for UTM accounting 199
WinAdmin 43, 123
administering with 123
Windows system 15
writing an administration program 115

X

XA interface
log calls (program) 376
xa_debug
kc_diag_and_account_par_str 614
KC_MODIFY_OBJECT 376
xa_debug_out
kc_diag_and_account_par_str 614
KC_MODIFY_OBJECT 376
XA-DEBUG
enabling (KDCDIAG) 688
XA-DEBUG-OUT
controlling XA-DEBUG (KDCDIAG) 688
XAPTP trace in test mode 685
XATMI
trace 372
XATMI trace
enabling/disabling 372
XATMI trace function 616
xatmi_trace
KC_MODIFY_OBJECT 372