

openUTM V6.1

Generating Applications

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 © Fujitsu Technology Solutions GmbH 2011.

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	14
1.2	Summary of contents of the openUTM documentation	15
1.2.1	openUTM documentation	15
1.2.2	Documentation for the openSEAS product environment	19
1.2.3	README files	20
1.3	Innovations in openUTM V6.1	21
1.3.1	New server functions	21
1.3.1.1	New functions operative in all UTM applications	21
1.3.1.2	New functions in UTM cluster applications	22
1.3.2	New client functions	25
1.3.3	New and modified functions for openUTM WinAdmin	26
1.4	Notational conventions	28
2	Introduction to the generation procedure	31
2.1	Configuring the UTM application	33
2.2	Generating application components - result of the KDCDEF run	34
2.3	The KDCFILE	45
2.3.1	Administrative data	49
2.3.2	Page pool	49
2.3.3	Restart area	52
2.3.4	Creating a new KDCFILE during operation	55
2.4	Performance aspects - tuning	57
2.4.1	Splitting the KDCFILE	57
2.4.2	KDCFILE on raw-device (Unix systems)	59
2.4.3	KDCFILE on a stripe set in (Windows systems)	62

3	Notes on generating a UTM cluster application	63
3.1	Generating a UTM cluster application	64
3.1.1	UTM cluster files	64
3.1.2	KDCDEF statements	69
3.1.3	Initial KDCFILE	70
3.2	Generating a reserve node application	70
3.3	Using global memory areas	71
3.4	Using users with RESTART=YES	71
3.5	Special issues in BS2000/OSD	72
3.6	Special issues on Unix systems and Windows systems	73
3.7	Special issues with LU6.1 connections	73
4	Generating applications for distributed processing	75
4.1	Distributed processing via the LU6.1 protocol	76
4.1.1	Transport connections and SNA sessions	76
4.1.2	Generation notes	78
4.1.3	Procedure when generating LU6.1 connections	81
4.1.4	LU6.1-LPAP bundles	88
4.1.5	LU6.1-LPAP bundles of a standalone application with a UTM cluster application	90
4.2	Distributed processing via the OSI TP protocol	93
4.2.1	OSI terms	93
4.2.2	Generation procedure for distributed processing based on OSI TP	99
4.2.3	OSI-LPAP bundles	105
4.3	Coordinating the UTM and BCAM generations (BS2000/OSD)	109
4.4	Providing address information for the CMX transport system (Unix systems and Windows systems)	110
4.4.1	Providing address information with KDCDEF	110
4.4.2	Providing address information with TNS entries	113
4.4.3	Converting address information from TNS entries to KDCDEF	114
4.5	Providing address information for the SOCKET transport system (Unix systems and Windows systems)	118
4.6	Single- and multi-threaded network access (Unix systems and Windows systems)	120

4.7	Using mapped host names (Unix systems and Window systems)	122
4.7.1	Conversion file for mapped host names	123
4.7.2	UTM_NET_HOSTNAME environment variable	125
5	Generating selected objects and functions of the application	127
<hr/>		
5.1	Connecting clients to the application	128
5.1.1	Connecting clients via LTERM partners	129
5.1.2	LTERM pools	133
5.1.3	LTERM bundle	137
5.1.4	LTERM groups	140
5.1.5	Connecting OpenCPIC clients	143
5.1.6	Defining the client sign-on services	144
5.1.6.1	Establishing an automatic connection	145
5.1.6.2	Automatic sign-on under a specific user ID	145
5.1.6.3	Generating sign-on services for clients	145
5.1.6.4	Multiple sign-ons	146
5.1.7	Specifying maximum waiting times for dialog prompting	147
5.1.8	Generating security functions	148
5.1.8.1	Defining system access control	148
5.1.8.2	Assigning administration authorizations	149
5.1.9	Generating a restart	150
5.1.10	USP headers for output messages to socket connections	151
5.1.11	Code conversion	152
5.1.12	Providing address information	153
5.1.12.1	Providing the address information for clients of type SOCKET	153
5.1.12.2	Providing address information for clients of type UPIC and APPLI in BS2000/OSD	155
5.1.12.3	Providing address information for clients of type UPIC and APPLI in Unix systems and Windows systems	156
5.1.12.4	Additional information for LTERM pools in Unix systems and Windows systems	159
5.1.12.5	Providing address information for OpenCPIC clients in BS2000/OSD	160
5.1.12.6	Providing address information for OpenCPIC clients in Unix systems and Windows systems	161
5.1.13	Examples of the generation of a client/server cluster	163
5.2	Generating printers (on BS2000/OSD and Unix systems)	168
5.2.1	Generating RSO printers (BS2000/OSD)	172
5.2.1.1	Entries for the KDCDEF generation	172
5.2.1.2	Entries for RSO and SPOOL	173
5.2.1.3	Activating printers for openUTM	177
5.2.1.4	Querying printer information	177
5.2.1.5	Releasing printers in the event of an error	178

5.2.2	Generating printer pools	178
5.2.3	Bypass mode (BS2000/OSD)	178
5.2.4	Generating printer control LTERMs	179
5.3	Generating service-controlled queues	181
5.3.1	USER queues	181
5.3.2	TAC queues	182
5.3.3	Temporary queues	184
5.3.4	Specifying the maximum waiting time for reading from service-controlled queues	185
5.3.5	Limiting the maximum number of redeliveries to service-controlled queues	185
5.4	UTM messages	186
5.4.1	Messages in openUTM under BS2000/OSD	186
5.4.2	Messages in openUTM under Unix systems and Windows systems	189
5.4.3	User-specific message destinations	191
5.5	Message distribution and multiplexing with OMNIS (BS2000/OSD)	192
5.5.1	Multiplex connections	193
5.5.1.1	Defining multiplex connections	194
5.5.1.2	Outputting asynchronous messages without prior announcement	196
5.5.1.3	Confirming the connection shutdown by the partner	196
5.5.2	Statistics on multiplex connections	197
5.5.3	Combination of multiplex connections and direct connections	197
5.6	Generating load modules, common memory pools and shared code (BS2000/OSD)	199
5.6.1	Generating load modules	199
5.6.2	Generating shared code and common memory pools	204
5.6.2.1	Shared code in system memory	204
5.6.2.2	Shared code in common memory pools	205
5.7	Job control - priorities and process limitations	208
5.7.1	Job processing via priority control	211
5.7.2	Job processing via process limitation for TAC classes control	214
5.7.3	Comparison of some of the properties of the two methods	215
5.7.4	Process priorities in BS2000/OSD	218
5.8	Data access control	219
5.8.1	Lock/key code concept	219
5.8.2	Access list concept	222
5.8.3	Data access control with distributed processing	226
5.9	Message encryption on connections to clients	228
5.9.1	Requirements	228
5.9.2	Encryption methods	229

5.9.3	Encrypting passwords and user data	230
5.9.3.1	System access control	231
5.9.3.2	Data access control	232
5.9.4	Creating the RSA key pair and reading the public key	233
5.10	Defining database linking	234
5.10.1	Linking databases under BS2000/OSD	234
5.10.2	Linking to a Resource Manager under Unix systems and Windows systems	235
5.11	Internationalizing the application – XHCS support (BS2000/OSD)	237
5.11.1	Definitions of XHCS terms	238
5.11.2	Defining the language environment – setting the locale	240
5.11.3	Character set names for edit profiles and formats	242
5.12	Generating system access control using Kerberos (BS2000/OSD)	243
6	The KDCDEF generation tool	245
6.1	Creating the ROOT table source and KDCFILE	245
6.1.1	Statements for controlling the KDCDEF run	246
6.1.2	Statements for creating the ROOT table source	246
6.1.3	Basic statements for creating a KDCFILE	247
6.1.3.1	Creating the KDCFILE - additional statements for distributed processing via LU6.1	248
6.1.3.2	Creating the KDCFILE - additional statements for distributed processing via OSI TP	249
6.1.3.3	Creating the KDCFILE - additional statements for UTM cluster applications	249
6.1.3.4	Statements used to generate the UTM cluster files	250
6.1.4	Effects of the KDCDEF statements on the generation objects	251
6.2	Calling KDCDEF and entering the control statements	254
6.2.1	Starting KDCDEF and executing a KDCDEF run	254
6.2.1.1	BS2000/OSD	254
6.2.1.2	Unix systems	255
6.2.1.3	Windows systems	256
6.2.2	Order of the control statements	257
6.2.3	Format of the control statements	258
6.2.4	Continuation lines in control statements	258
6.2.5	Syntax and plausibility checks	259
6.2.6	KDCDEF logging	259

6.2.7	Format and uniqueness of object names	260
6.2.7.1	Reserved names	260
6.2.7.2	Format of names	260
6.2.7.3	Number of names	262
6.2.7.4	Uniqueness of names and addresses	265
6.2.8	Result of the KDCDEF run	267
6.3	Inverse KDCDEF	268
6.3.1	Starting inverse KDCDEF	270
6.3.2	Result of inverse KDCDEF	271
6.3.3	Creating KDCDEF control statements in upgrades	273
6.4	Recommendations when regenerating an application	273
6.5	KDCDEF control statements	275
	ABSTRACT-SYNTAX - define the abstract syntax	275
	ACCESS-POINT - create an OSI TP access point	277
	ACCOUNT - define the accounting functions	283
	APPLICATION-CONTEXT - define the application context	285
	AREA - define additional data areas	288
	BCAMAPPL - define additional application names	291
	CLUSTER – Define global properties of a UTM cluster application	299
	CLUSTER-NODE – Define a node application of a UTM cluster application	310
	CON - define a connection for distributed processing based on LU6.1	313
	CREATE-CONTROL-STATEMENTS - Create KDCDEF control statements	318
	DATABASE - define the database system (BS2000/OSD)	321
	DEFAULT - define default values (BS2000/OSD)	324
	EDIT - define edit options (BS2000/OSD)	328
	EJECT - initiate a page feed in the log	332
	END - terminate KDCDEF input	333
	EXIT - define event exits	334
	FORMSYS - define the format handling system	336
	KSET - define a key set	337
	LOAD-MODULE - define a load module (BLS, BS2000/OSD)	339
	LPAP - define an LPAP partner for distributed processing based on LU6.1	344
	LSES - define a session name for distributed processing based on LU6.1	347
	LTAC - define a transaction code for the partner application	348
	LTERM - define an LTERM partner for a client or printer	355
	MASTER-LU61-LPAP – Define the master LPAP of an LU6.1-LPAP bundle	366
	MASTER-OSI-LPAP - Defining the master LPAP of an OSI-LPAP bundle	367
	MAX - define UTM application parameters	368
	MESSAGE - define a UTM message module	406
	MPOOL - define a common memory pool (BS2000/OSD)	410
	MSG-DEST - define user-specific messages destinations	412
	MUX - define a multiplex connection (BS2000/OSD)	414

OPTION - manage the KDCDEF run	416
OSI-CON - define a logical connection to an OSI TP partner	420
OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP	426
PROGRAM - define a program unit	434
PTERM - define the properties of a client/printer and assign an LTERM partner	437
QUEUE - reserve table entries for temporary messages queues	458
REMARK - insert a comment line	460
RESERVE - reserve table locations for UTM objects	461
RMXA - define a name for an XA (database) connection (Unix systems, Windows systems)	466
ROOT - define a name for the ROOT table source	467
SATSEL - define SAT logging (BS2000/OSD)	468
SESCHA - define session characteristics for distributed processing based on LU6.1	470
SFUNC - define function keys	473
SHARED-OBJECT - define shared objects/DLLs (Unix systems, Windows systems)	476
SIGNON - control the sign-on procedure	478
TAC - define the properties of transaction codes and TAC queues	483
TACCLASS - define the number of processes for a TAC class	500
TAC-PRIORITIES - specify priorities of the TAC classes	505
TCBENTRY - define a group of TCB entries (BS2000/OSD)	509
TLS - define a name for a TLS block	510
TPOOL - define an LTERM pool	511
TRANSFER-SYNTAX - define the transfer syntax	528
ULS - define a name for a ULS block	529
USER - define a user ID	530
UTMD - application parameters for distributed processing	544
6.6	Dialog control - effects of generation parameters 548
6.7	Example generation: <i>Comfo</i> TRAVEL 550
6.7.1	KDCDEF input file DYNAMIC.RMS for UTM-D application RMS 551
6.7.2	KDCDEF statements for UTM-D application RMS 554
6.7.3	KDCDEF input file DynamicTravel for UTM application TRAVEL 560
6.7.4	KDCDEF statements for UTM application TRAVEL 562
6.8	KDCDEF messages 567
7	Changing the configuration of an application dynamically 569
7.1	Reserving locations in the KDCFILE object tables 571
7.2	Prerequisites for entering objects dynamically 573

8	The tool KDCUPD – updating the KDCFILE	577
8.1	Overview	578
8.1.1	Supported upgrades	578
8.1.2	Prerequisite for using KDCUPD	578
8.1.3	Backing up data	579
8.1.4	What data does KDCUPD transfer?	580
8.1.4.1	Changing generation parameters	582
8.1.4.2	Transfer of user data	583
8.2	Updating the KDCFILE for standalone UTM applications	585
8.3	Updating the KDCFILE and UTM cluster files for UTM cluster applications	590
8.3.1	Online update of a UTM cluster application	591
8.3.2	Update generation for a UTM cluster application	592
8.3.3	Converting a UTM cluster application	594
8.3.3.1	Conversion from a standalone UTM application to a UTM cluster application	594
8.3.3.2	Converting a UTM cluster application from V6.0 to V6.1	596
8.3.3.3	Converting a UTM cluster application to a standalone UTM application	597
8.4	Control statements for KDCUPD	600
	CATID - define Catid of the old and the new KDCFILE	601
	CHECK - check the consistency of the KDCFILE	601
	CLUSTER-FILEBASE - Specify the base names of the old and new UTM cluster files	602
	END - terminate input and start processing	602
	KDCFILE - specify the base name of the old and new KDCFILE	603
	LIST - control the runtime log	604
	TRANSFER - control the data transfer of the user data	606
8.5	KDCUPD runtime log and messages	610
9	Appendix	613
9.1	Code conversion tables in BS2000/OSD	613
9.2	Code conversion tables in Unix systems and Windows systems	614
9.2.1	Modifying the code table in Unix systems	614
9.2.2	Modifying the code table in Windows systems	615

Glossary 617

Abbreviations 651

Related publications 657

Index 667

1 Preface

Modern enterprise-wide IT environments are subjected to many challenges of an increasingly explosive nature. This is the result of:

- heterogeneous system landscapes
- different hardware platforms
- different networks and different types of network access (TCP/IP, SNA, HTTP)
- 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 transaction-oriented middleware platform 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 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 Oracle/Sun's Java Connector Architecture (JCA) and supports standardized connection of UTM applications to J2EE 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 openUTM manual “Generating Applications” is designed for use by application planners and developers as well as operators of UTM applications.

This manual describes how to define the configuration for a UTM application using the UTM tool KDCDEF and how to create the KDCFILE. Chapter 5 also goes into more detail about the generation of selected objects and functions of the application.

Additional topics include the dynamic configuration of an application and the updating of the KDCFILE using the tool KDCUPD.

To understand this manual you will need to be familiar with the operating system.



Wherever the term Unix system or Unix platform is used in the following, then this should be understood to mean both a Unix-based operating system such as Solaris or HP-UX and 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, an online help system for openUTM WinAdmin, which is the graphical administration workstation, 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/OSD, Unix systems or 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 V6.1” in the **Search by product** field to display all openUTM manuals of version 6.1.

The manuals are included on the Enterprise DVD with open platforms and are also available on the WinAdmin DVD (for BS2000/OSD).

The following sections provide a task-oriented overview of the openUTM V6.1 documentation. You will find a complete list of documentation for openUTM in the chapter on related publications at the back of the manual on [page 657](#).

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/OSD, Unix based 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/OSD there is supplementary documentation on the programming languages Assembler, Fortran, Pascal-XT and PL/1. T

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 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/OSD or Unix systems/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 **openUTM WinAdmin**, the graphical administration workstation, the following documentation is available to you:
 - A **description of WinAdmin**, which provides a comprehensive overview of the functional scope and handling of WinAdmin. This document is shipped with the software and is also available online as a PDF file.
 - The **online help system**, which provides 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 in order to administer standalone UTM applications and UTM cluster applications.

Testing and diagnosing errors

You will also require the **Messages, Debugging and Diagnostics** manuals (there are separate manuals for Unix systems / Windows systems and for BS2000/OSD) 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.

1.2.2 Documentation for the openSEAS product environment

This manual briefly describes how openUTM is connected to the openSEAS product environment in chapter 2. The following sections indicate which openSEAS documentation is relevant to openUTM.

Integrating J2EE 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 J2EE 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-compliant adapter which connects UTM applications with applications based on J2EE, 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 online

Readme files are available to you online in addition to the product manuals under the various products at <http://manuals.ts.fujitsu.com>.

Readme files under BS2000/OSD

On your BS2000 system you will find Readme files for the installed products under the file name:

```
SYSRME.<product>.<version>.E
```

Please refer to your system administrator for the user ID under which the required Readme file can be found. You can also obtain the path name of the Readme file directly by entering the following command:

```
/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>, LOGICAL-ID=SYSRME.E
```

You can view the Readme file on screen with `/SHOW-FILE` or by opening it in an editor, or print it on a standard printer using the following command:

```
/PRINT-DOCUMENT <filename>, LINE-SPACING=*BY-EBCDIC-CONTROL
```

Readme files under Unix 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 under 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 V6.1

The functionality of UTM cluster applications has been greatly extended. In addition, openUTM offers a number of new server-side and client-side functions. The graphical administration workstation WinAdmin has been converted to Java technology and has also been extended to include new functions.

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

1.3.1 New server functions

Some of the enhancements work in all UTM applications whereas others can only be used in UTM cluster applications.

1.3.1.1 New functions operative in all UTM applications

The functions listed below work in both standalone applications and in UTM cluster applications.

Generation

- New default value for MAX_REQNR in BS2000. For performance reasons, this has been increased from 2 to 20.

KDCADMI enhancements

The following new and modified object types, operation codes and data structures are now available for administration tasks:

- New object type KC_PTC for the operation code KC_GET_OBJECT with the data structure *kc_ptc_str* for the display of distributed transactions with the state PTC (Prepare to Commit).
- New operation code KC_PTC_TA with sub-opcode KC_ROLLBACK for the rollback of transactions with the state PTC.
- New object type KC_DB_INFO for operation code KC_GET_OBJECT with new data structure *kc_db_info_str* for the output of information on the database connection.
- New sub-opcode KC_SAME in the operation code KC_CHANGE_APPLICATION to make it possible to reload the application program on open platforms without it being necessary to use a new version of the program when FGs are employed.

- When the operation code `KC_GET_OBJECT` is used with the object type `KC_USER` or `KC_USER_DYN2`, the system also indicates whether the user has an open service with a transaction in the state `PTC`.
- Data structure `kc_tac_str`
In the case of an XA connection, the fields `db_counter` and `db_elap_msec` no longer return the value 0 but instead binary zero since the database values cannot be captured in the case of an XA connection.

Extensions to commands

- The command `KDCAPPL PROG=SAME` can be used to reload the application program on open platforms without it being necessary to use a new version of the program when FGGs are employed (see also the `KDCADMI` program interface).
- The output from `KDCINF SYSPARM` now contains additional information.

1.3.1.2 New functions in UTM cluster applications

Functional enhancements for UTM cluster applications

- The global storage areas `GSSB` and `ULS` can be used globally in the cluster

`GSSB` and `ULS` are administered in such a way that they are available for all users in all node applications. I.e. all node applications have the same view of the data in the storage area. If a storage area is set up (`GSSB`) or modified (`GSSB`, `ULS`) in a node application then this is also visible to all the other node applications in this UTM cluster application after the transaction has ended.

To permit the use of `GSSB` and `ULS` globally in the cluster there is a special lock management function for which an optional deadlock detection mechanism has also been implemented.
- The dialog services of users generated with `RESTART=YES` are valid globally in the cluster.

This means that users who were generated with `RESTART=YES` can usually continue open transactions at another node application.
- `KDCUPD`

The `KDCUPD` tool now has two modes for UTM cluster applications:

 - A mode in which the user data that is local to the node is transferred for each node application
 - A mode in which the user data that is valid globally in the cluster (`ULS`, `GSSB`, service data, passwords, locales) is transferred

- Cluster-internal communication

Cluster-internal communication has been modified in order to improve performance.

Additional UTM cluster files

Additional files are required for the new functionality available in UTM cluster applications:

- The cluster GSSB file and cluster ULS file are created at generation time in order to permit the global administration of the GSSB and ULS areas throughout the cluster.
- To permit lock management globally throughout the cluster, there is one separate lock management file. This is created as soon as the UTM cluster application is started with the newly generated UTM cluster files.
- A cluster page pool for the storage of the user data applicable throughout the cluster is created at generation time. This user data includes, for example, the contents of GSSB and ULS and the data of a user's dialog service that has been generated with `RESTART=YES`.

Note: There is still also a local page pool for each node application. This is used to store the user data that is local to the node.

Generation

The following statements have been modified in order to make it possible to generate UTM cluster applications:

- **CLUSTER**
 - New operands `ABORT-BOUND-SERVICE`, `DEADLOCK-PREVENTION`, `PGPOOL` and `PGPOOLFS` which make it possible to control service restarts, the locking behavior for global memory areas and the properties of the cluster page pool.
 - The permitted range of values for the `FILE-LOCK-RETRY` and `FILE-LOCK-TIMER-SEC` operands has been changed.
 - The operand `LISTENER-ID` is also now available for Unix systems and Windows systems.
 - The `USER-RESTART` and `GLOBAL-UTM-DATA` operands are no longer supported.
 - The `LISTENER-PORT` operand is no longer supported under BS2000.
- **CLUSTER-NODE**
 - The length of the `HOSTNAME` operand is now limited to a maximum of 8 characters.
 - `catid_B` is no longer supported in BS2000/OSD.

- MAX
SINGLE is the only permitted specification for the KDCFILE operand.
- LTERM and USER
The default value is RESTART=YES exactly as in the case of standalone applications.
- OPTION
If GEN=CLUSTER is set then the UTM cluster files that are set up on generation are always regenerated. They must not exist prior to this.

Enhancements to KDCADMI for UTM cluster applications

The following new and modified object types, operation codes and data structures are now available for the administration of UTM cluster applications:

- New object type KC_CLUSTER_CURR_PAR for the operation codes KC_GET_OBJECT and KC_MODIFY_OBJECT with data structure *kc_cluster_curr_par_str* in order to display the current values of UTM cluster applications, for example the cluster page pool allocation.
- The operation code KC_LOCK_MGMT now has the new sub-opcodes KC_ABORT_BOUND_SERVICE, KC_ABORT_ALL_BOUND_SERVICES and KC_ABORT_PTC_SERVICE.

These sub-opcodes are used to mark one or more (user) services bound to an abnormally terminated node application for abnormal termination. I.e. the services are terminated abnormally immediately on the start-up of the node application to which they are bound.

This enables users to sign on at another node application.
- Using the KC_GET_OBJECT with the object type KC_USER or KC_USER_DYN2 now also shows whether the user
 - has a node-bound service
 - or a node-bound service with a transaction in the state PTC.
- New sub-opcode KC_READ_NO_GSSBFILE to specify whether or not the cluster GSSB file should be accessed when reading (further) GSSB objects with KC_GET_OBJECT. Using this sub-opcode for follow-up calls at KDCADMI considerably improves performance when a large number of GSSBs are present.
- Data structure *kc_curr_par_str* (current values of the user parameters)
kc_curr_par_str has been extended in order to display statistics fields relating to lock conflicts.

- Data structure *kc_cluster_par_str* (properties of a UTM cluster application)
 - *kc_cluster_par_str* has been extended to make it possible to display the new settings of UTM cluster applications (e.g. sign-on behavior in the case of open, node-bound services, cluster page pool) and control the locking behavior for global storage areas.
 - The structure no longer contains the fields *global_utm_data* and *user_restart*.
- Data structure *kc_cluster_node_str*

The *hostname* field has been shortened to a length of 64 characters.
- Data structure *kc_lock_mgmt_str*

The name *kc_lock_mgmt_str* given to the structure used to release locks in V6.0 has been changed to *kc_lock_mgmt_str* in V6.1.

Messages

The messages K168, K172, K173, K177 and K187 present in UTM V6.0 have been replaced by the new message K190.

1.3.2 New client functions

Some of the changes apply to all clients, i.e. they apply both to the clients of standalone UTM applications and those of UTM cluster applications. Other changes apply only to the clients of UTM cluster applications.

New and modified functions for all UPIC clients

- WARN or GRACE shutdown in a UTM application

UPIC client applications can react specifically to WARN or GRACE shutdowns. If a connection has been established to a UTM application and a WARN or GRACE shutdown is then triggered in the application, this is passed on to the client via the UPIC protocol. The client program can query the shutdown status and shutdown time using the new functions *Extract_Shutdown_State* and *Extract_Shutdown_Time*.
- Elimination of the OCX interface

The OCX interface is no longer provided in UPIC. Consequently, the section "ActiveX Control UpicB.ocx" is no longer included.
- Linux x86 64-bit platform

UPIC client applications can run in either a 32-bit or 64-bit environment on Linux x86 systems.

New functions for UPIC clients of UTM cluster applications

This functionality is only available if upicfile cluster entries (prefix CD) are used.

- WARN or GRACE shutdown in a UTM cluster application

If, after connecting to a node application, the UPIC client detects that the node application is already in a WARN or GRACE shutdown state, then it disconnects again and establishes a connection to another node application.
- Changing node application while a service is open

If a user sign-on is rejected at a node application because a service that is bound to another node application is open for this user, then UPIC attempts to sign on this user at this (other) node application.

1.3.3 New and modified functions for openUTM WinAdmin

Not only has WinAdmin been converted to Java technology, it also possesses a range of new and modified functions. Some of these functions are available for all applications (standalone UTM applications and UTM cluster applications) whereas others apply only to UTM cluster applications.

Changeover to Java technology

WinAdmin has been converted to Java technology:

- Menu bar

The menu bar has been modified and simplified. The *Application*, *Cluster*, *Edit*, *Position*, *Queue* and *System* menus are no longer present. The functionality of these menus has been included in the context menus.
- List windows and text windows

The display has been simplified. At the same time, the functionality has been greatly extended. The toolbar present in earlier versions is no longer present.
- Tooltips in object lists and property dialogs

Tooltips are displayed in UTM object lists and the property dialogs of UTM objects. These provide more detailed information on the cell contents and object properties. This is particularly beneficial in the case of UTM cluster applications since it means that the values of the individual node applications are available in lists and dialogs that apply globally throughout the cluster.
- Definition of partial object lists
- The online Help system supplied with WinAdmin runs as a JavaHelp system.

- Performance

Performance has been considerably improved through the use of parallel UPIC connections and asynchronous communications. Among other things, WinAdmin therefore now reacts faster to input entered in parallel to communications activities.

New functions operative in all UTM applications

- Display of all open services in a running UTM application that have a transaction with the state PTC.
- Display of the generated database systems
- Function for rolling back transactions with the state PTC
- Display of the applications that are present in a collection

New functions operative in UTM cluster applications

- Display of the new parameters for a UTM cluster application
- Display of the current values for a UTM cluster application
- New functions at the node application in order to mark all services bound to an abnormally terminated node application for abnormal termination so that the users can sign on at another node application.
- New functions for the **User** object in order to mark the services associated with users for abnormal termination:
 - Services bound to a node application that has terminated abnormally
 - Services with a transaction with the state PTC that are bound to a node application that has terminated abnormally

This allows the user to sign on at another node application.

- Activation/deactivation of deadlock detection in UTM cluster applications

Elimination of the Generate function

The Generate function is no longer available in WinAdmin. As a result, the entire associated **Generate** subtree is no longer displayed and all the related commands are no longer present in the context menus.

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
<i>lowercase letters in italics</i>	In running text, variables and the names of data structures and fields are indicated by lowercase letters in italics.	<i>filebase</i> is the basic name of KDCFILE
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 omitted.	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,...keyn)
<u>Underscoring</u>	Underscoring denotes the default value.	CONNECT= { A/YES <u>NO</u> }

Representation	Meaning	Example
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.	TRANSPORT- SELECTOR =c'c'
...	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/OSD-specific elements of a description.
- X**
X This symbol is used in the left-hand margin to indicate Unix 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 in BS2000/OSD and Unix 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 in BS2000/OSD 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 in Unix systems and Windows systems.



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



Indicates notes that are of particular importance.



Indicates warnings.

2 Introduction to the generation procedure



In the form presented here, this chapter applies only to standalone applications. In the case of UTM cluster applications, there are, on the one hand, a number of additional files (the so-called UTM cluster files) and, on the other, a series of special considerations relating, for example, to the properties of the KDCFILE or to the KDCUPD. For further details, see [chapter “Notes on generating a UTM cluster application” on page 63](#)

Alongside the program units that provide the services and the formats (for formatted operation in BS2000/OSD) you must create the following application components for a UTM application:

- **KDCFILE configuration file**
The KDCFILE contains the configuration of your application. openUTM stores all the administrative data required to operate the application in the KDCFILE and reserves areas for the user data and for transaction management. When operating the application, all tasks and work processes of the application access the KDCFILE. This manual describes how to create the KDCFILE.
- **KDCROOT main routine**
The program units you have created run under the control of KDCROOT. The ROOT tables are used as the basis of the main routine KDCROOT. The ROOT tables contain application-specific configuration data that is required by the main routine KDCROOT. When operating an application, the main routine KDCROOT establishes the connection from openUTM to the program units, the database and under BS2000/OSD to the formatting system. This manual also describes how to create the sources for the ROOT tables.

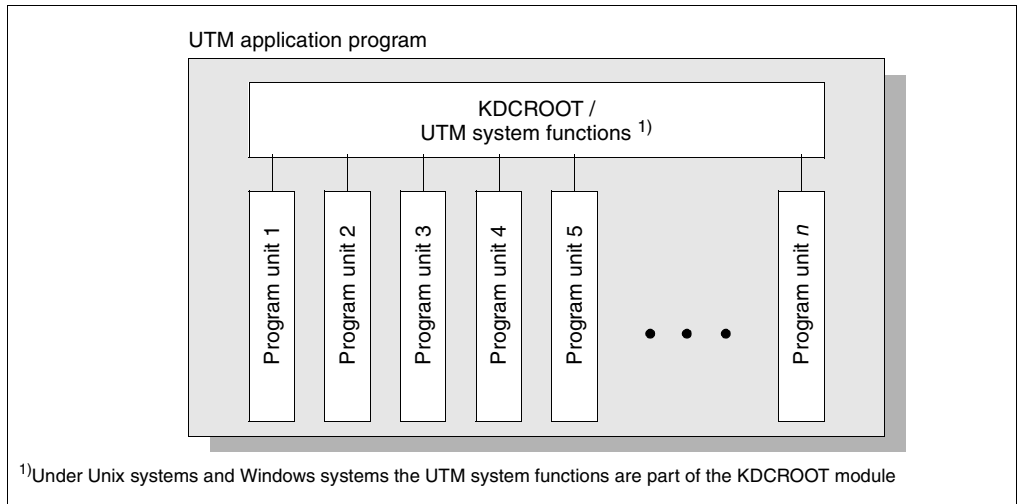


Figure 1: Structure of the UTM application program

In order to create the ROOT table source and the KDCFILE, you must first define the configuration of application. This entire procedure is known as “generation”. To allow you to configure and generate the KDCFILE and the ROOT table sources, openUTM provides the KDCDEF generation tool.



The KDCDEF generation tool is described in detail in [chapter “The KDCDEF generation tool” on page 245](#).

Information on the KDCFILE can be found on [page 45](#).

The information on generation contained in this [chapter “Introduction to the generation procedure”](#) applies both to standalone UTM applications and to UTM cluster applications. You will find additional information on generating UTM cluster applications in the [chapter “Notes on generating a UTM cluster application” on page 63](#).

You must create the application program using KDCROOT, user program units, interfaces and other application components like the UTM system modules, the runtime systems of the programming languages, database connection modules etc.



More information about creating application programs using ROOT tables and application components can be found in the corresponding openUTM manual “Using openUTM Applications”.

Information about creating application program units can be found in the openUTM manuals “Programming Applications with KDCS” and “Creating Applications with X/Open Interfaces”.

B
B

Information about creating formats in BS200/OSD can be found in the manuals for FHS.

2.1 Configuring the UTM application

To execute the application program, you must define the following information for example:

- the application properties
- the UTM user IDs and data access control
- the properties of clients and printers
- the properties of partner applications (server applications)
- the properties of services, i.e. of transaction codes and program units
- message queues (user, TAC and temporary queues)
- the structure of the application (subdivision into load modules for use with BLS, shared objects or DLLs)
- reserved locations in UTM object tables for dynamic configuration

These properties combine to form the configuration, and are defined using the KDCDEF control statements. The KDCDEF control statements serve as input for the generation tool KDCDEF.

The KDCDEF control statements are listed in accordance with their function group starting on [page 245](#).

The KDCFILE administrative file is used to store all configuration information and thus all administrative data required to run the application.

2.2 Generating application components - result of the KDCDEF run

You can generate the KDCFILE and the ROOT table sources in a single KDCDEF run or in separate KDCDEF runs. The KDCDEF statement OPTION allows you to define the generation objects to be created by KDCDEF:

```
OPTION...,GEN={ KDCFILE | ROOTSRC | NO | ALL }
```

The name of the ROOT tables is defined using the ROOT statement.

```
ROOT rootname
```

B Under BS2000/OSD *rootname* is the name of the ROOT table module.

X/W
X/W Under Unix systems and Windows systems *rootname* is a name component of the ROOT table source (ROOTSRC).

KDCDEF reads the control statements from standard input or from a file.

B Under BS2000/OSD standard input means SYSDTA (with the SDF command /ASSIGN-SYSDTA you can assign SYSDTA to a SAM or ISAM file, a library member of type S, a PLAM library, or *SYSCMD, for example)

X/W
X/W Under Unix systems and Windows systems standard input means *stdin* (i.e. from the Unix or DOS command level).

You will find a detailed description of how to start KDCDEF and pass the control statements to KDCDEF on [page 254](#).

All KDCDEF statements are subjected to syntax and plausibility checks. If KDCDEF does not detect any serious errors in this process, the files listed in [figure 2 on page 35](#) are created for a standalone UTM application.

[Figure 5 on page 66](#) shows what files are created when you generate a UTM cluster application.



Even if the OPTION statement is used in a KDCDEF run to cause only part of the configuration to be (newly) created, you nevertheless specify the statements for the entire configuration for every generation run. Only then is KDCDEF able to check the completeness and consistency of the generation statements.

KDCDEF always performs plausibility checks for all statements. If, for instance, only a ROOT source is to be generated in a KDCDEF run, KDCDEF also checks the statements that only affect the KDCFILE.

This complete check allows inconsistencies that arise on creating the ROOT table module and the KDCFILE that would otherwise only be detected when the application is started to be identified early and consequential errors to be avoided.

Figure 2 shows what files are created when you define a standalone UTM application.

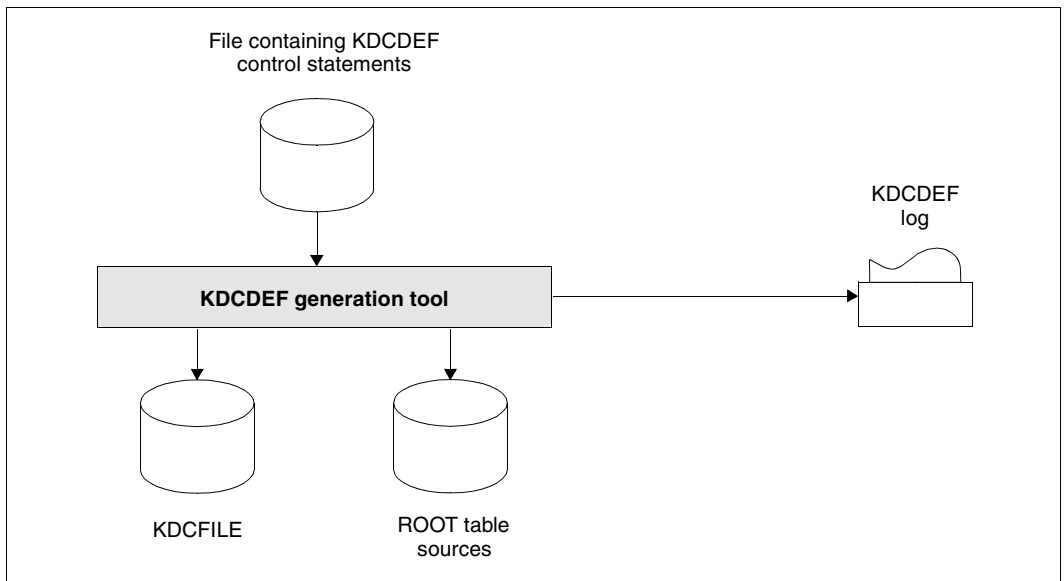


Figure 2: The result of the KDCDEF run (with OPTION ...,GEN=ALL) for a standalone UTM application.



It is strongly recommended to carry out every generation run with all generation information, regardless of whether just the source code for the ROOT tables or just the KDCFILE is to be created.

KDCDEF always executes plausibility checks for all statements. If only one ROOT source is to be created in a KDCDEF run, for example, then KDCDEF also checks the statements that only affect the KDCFILE.

Inconsistencies that may arise while creating the ROOT table modules and KDCFILE that would usually only be detected once the application is started can be detected earlier by this complete check, thereby reducing secondary errors.

KDCDEF statements for a minimal configuration

You must pass at least the following control statements to KDCDEF before you can run your UTM application.

You must execute additional KDCDEF statements for distributed processing, connecting printers, etc. You will find more information on this subject in [section “Distributed processing via the LU6.1 protocol” on page 76](#), [section “Distributed processing via the OSI TP protocol” on page 93](#) and [section “Generating printers \(on BS2000/OSD and Unix systems\)” on page 168](#).

The lines indicated by '*' are comments.

B *Minimal configuration for BS2000/OSD*

```

B *****
B * Specify which part of the application program is to be created          *
B * by KDCDEF                                                                *
B *****
B OPTION GEN=...
B
B *****
B * Specify the name of the Root table                                       *
B *****
B ROOT applroot
B
B *****
B * Specify application parameters                                           *
B *****
B * Application name with which the application is started or with which
B *   clients can address the application
B
B MAX APPLNAME=sample
B
B * Specify the base directory of the application.
B
B *MAX KDCFILE=filebase
B
B * Define the maximum number of process of the UTM application:
B
B MAX TASKS=2

```

```

B *****
B * optional: Generate the database system (ORACLE in the example) *
B *****
B * BS2000/OSD:
B DATABASE TYPE=XA
B
B *****
B * optional: Specify the formatting system used *
B *****
B * the statement FORMSYS must only be specified if your UTM application
B * is to run in formatted mode
B FORMSYS TYPE=FHS
B
B *****
B * optional: Connection points (LTERM partners) for clients/TS applications *
B *****
B * For example, generate open LTERM pools so that clients/TS applications
B * can connect to the application
B * LTERM pools for the various types of client ----- (1)
B TPOOL LTERM=client,NUMBER=...,PTYPE=*ANY,PRONAM=*ANY
B TPOOL LTERM=clientr,NUMBER=...,PTYPE=UPIC-R,PRONAM=*ANY
B TPOOL LTERM=appli,NUMBER=...,PTYPE=APPLI,PRONAM=*ANY
B TPOOL LTERM=socket,NUMBER=...,PTYPE=SOCKET,PRONAM=*ANY
B
B *****
B * Generate services *
B *****
B * Some own program units that initiate services and generate the
B * corresponding transaction codes
B * (for COMP=... enter the compiler used or the runtime system, mostly
B * „ILCS“,)
B PROGRAM=userpu,COMP=...
B TAC usertc,PROGRAM=userpu. ...----- (2)
B

```

```

B *****
B * optional: Administration *
B *****
B * Administration program KDCADM
B PROGRAM KDCADM,COMP=ILCS )
B * Generate administration command KDCSHUT so that the application
B * can always be terminated normally
B TAC KDCSHUT,PROGRAM=KDCADM ... ----- (3)
B * In applications with user IDs:
B * user ID for the administrator
B USER admin,PERMIT=ADMIN,PASS=...
B * If administration will be done via openUTM-WinAdmin,
B * then you need to submit the following TAC and PROGRAM statements
B * and generate a connection for the UPIC client (an LTERM pool in this case)
B * Furthermore, you should then generate the admin user ID with administration
B * authorization and without the restart property or generate your own
B * user ID with administration authorization and without the restart property
B * for administration using openUTM WinAdmin.
B Administration program KDCWADMI
B PROGRAM KDCWADMI,COMP=..
B TAC KDCWADMI,PROGRAM=KDCWADMI,CALL=BOTH,ADMIN=Y
B TPOOL LTERM=WADM,PTYPE=UPIC-R, PRONAM=*ANY, NUMBER=1
B
B *****
B *optional: Reserve space in the table for dynamic administration *
B *****
B RESERVE OBJECT=...,NUMBER=... ----- (4)
B
B END
B

```

B *Comments*

B (1) For each of the client types (terminal, UPIC client, TS application) that are to connect to the application, you must generate a separate LTERM pool. For terminals, a single LTERM pool is sufficient - depending on the type of terminals that are to sign in to the application. You can also generate the LTERM pools so that all clients of a particular type can log in - regardless of the computer on which they are located.

B You can also implement client connections with the help of the LTERM/PTERM statements. In particular, you must use LTERM/PTERM statements if the UTM application itself establishes connections to clients (e.g. TS applications) or if a printer is to be generated.

B (2) You can also assign several transaction codes to a program unit if the program unit performs several different services.

B (3) You can generate all administration commands that you will want to use in operation using additional TAC statements. If you want to use your own administration programs for administration purposes, then you must generate these programs with the corresponding PROGRAM and TAC statements.

B (4) You can add additional objects to the application configuration during live operation with the help of the administration (see the openUTM manual "Administering Applications"). You will need to create space in the tables in the KDCFILE for these objects in the KDCDEF generation.

X/W *Minimal configuration for Unix systems, Windows systems*

```

X/W *****
X/W * Specify which part of the application program is to be created *
X/W * by KDCDEF *
X/W *****

X/W OPTION GEN=...
X/W

X/W *****
X/W * Specify the name of the Root table *
X/W *****

X/W ROOT applroot
X/W

X/W *****
X/W * Specify application parameters *
X/W *****

X/W * Application name with which the application is started or with which
X/W * clients can address the application

X/W MAX APPLNAME=sample

X/W * Specify the base directory of the application.
X/W * This directory is the directory in which the KDCFILE is stored,
X/W * amongst other things.

X/W MAX KDCFILE=filebase

X/W * Specify the key for the shared memory area

X/W MAX CACHESHMKEY=...,IPCshmKEY=...,KAASHMKEY=...
X/W [ ,OSISHMKEY=...,XAPTPshmKEY=...] ----- (1)
X/W

X/W Define the semaphore key for the global application semaphore
X/W MAX SEMARRAY=number, number1
X/W

X/W * Define the maximum number of process of the UTM application.
X/W
X/W MAX TASKS=2

X/W *****
X/W * optional: Generate the database system (in the example ORACLE) *
X/W *****

X/W RMXA XASWITCH=xaoswd,SPEC=C
X/W

```



```

X/W *****
X/W * optional: Connection points (LTERM partners) for clients/TS applications *
X/W *****
X/W * For example, generate open LTERM pools so that clients/TS applications
X/W * can connect to the application
X/W * LTERM pools for the various client types ----- (2)
X/W TPOOL LTERM=CLIENTR,PTYPE=UPIC-R,NUMBER=...
X/W TPOOL LTERM=CLIENTL,PTYPE=UPIC-L,NUMBER=...
X/W TPOOL LTERM=APPLI,PTYPE=APPLI,NUMBER=...
X/W TPOOL LTERM=SOCKET,PTYPE=SOCKET,NUMBER=...
X/W TPOOL LTERM=TERM,PTYPE=TTY,NUMBER=...
X/W
X/W *****
X/W * Generate services *
X/W *****
X/W * Some own program units that initiate services and generate the
X/W * corresponding transaction codes
X/W * (for COMP=... enter the compiler used)
X/W PROGRAM=userpu,COMP=...
X/W TAC usertc,PROGRAM=userpu. ...----- (3)
X/W

```

```

X/W *****
X/W * optional: Administration *
X/W *****
X/W * Administration program KDCADM
X/W PROGRAM KDCADM,COMP=C
X/W * Generate administration command KDCSHUT so that the application
X/W * can always be terminated normally
X/W TAC KDCSHUT,PROGRAM=KDCADM ...----- (4)
X/W * In applications with user IDs: user ID for the administrator
X/W USER admin,PERMIT=ADMIN,PASS=....
X/W * If administration will be done via openUTM-WinAdmin,
X/W * then you need to submit the following TAC and PROGRAM statements
X/W * and generate a connection for the UPIC client (an LTERM pool in this case)
X/W * Administration program KDCWADMI
X/W PROGRAM KDCWADMI,COMP=C
X/W TAC KDCWADMI,PROGRAM=KDCWADMI,CALL=BOTH,ADMIN=Y
X/W TPOOL LTERM=WADM,PTYPE=UPIC-R, NUMBER=1
X/W
X/W *****
X/W * optional: Reserve space in the table for dynamic administration *
X/W *****
X/W RESERVE OBJECT=...,NUMBER=... ----- (5)
X/W END
X/W

```

X/W *Comments*

- X/W (1) You only need to specify the shared memory key OSISHMKEY= and XAPTPSHMKEY= if you generate objects for communication via OSI TP. The other shared memory areas are needed by every UTM application, running under Unix systems or Windows.
- X/W (2) Under Unix systems or Windows systems you must generate a separate LTERM pool for each type of client (terminal, UPIC client, TS application) that is to be able to connect to the application. You can generate the LTERM pools so that all clients of a particular type are able to sign on - regardless of the computer on which they are located.

- X/W You can also implement client connections with the help of the LTERM/PTERM statements. In particular, you must use LTERM/PTERM statements if the UTM application itself establishes connections to clients (e.g. TS applications) or if a printer is to be generated on Unix systems.
- X/W X/W X/W X/W
- (3) You can also assign several transaction codes to a program unit if the program unit performs several different services.
- X/W (4) You can generate all administration commands that you will want to use in operation using additional TAC statements. If you want to use your own administration programs for administration purposes, then you must generate these programs with the corresponding PROGRAM and TAC statements.
- X/W X/W X/W X/W
- (5) You can add additional objects to the application configuration during live operation with the help of the administration (see the openUTM manual “Administering Applications”). You will need to create space in the tables in the KDCFILE for these objects in the KDCDEF generation.

Regenerating existing UTM applications

If you want to generate a new ROOT table source and/or a new KDCFILE for an existing application (i.e. KDCROOT and KDCFILE already exist), then you must note the following:

You must enter the information on objects that are entered dynamically in the KDCFILE during operation or whose properties have been changed in the new KDCFILE. The “inverse KDCDEF” function is provided for this purpose. With this function you create the control statements from the configuration information of the current KDCFILE that can be used immediately. You will need to call the CREATE-CONTROL-STATEMENTS control statement in the KDCDEF run in order to do this.

Via the UTM administration you can also execute the inverse KDCDEF run while the application is running.



You will find more information on the “inverse KDCDEF” function in [section “Inverse KDCDEF” on page 268](#).

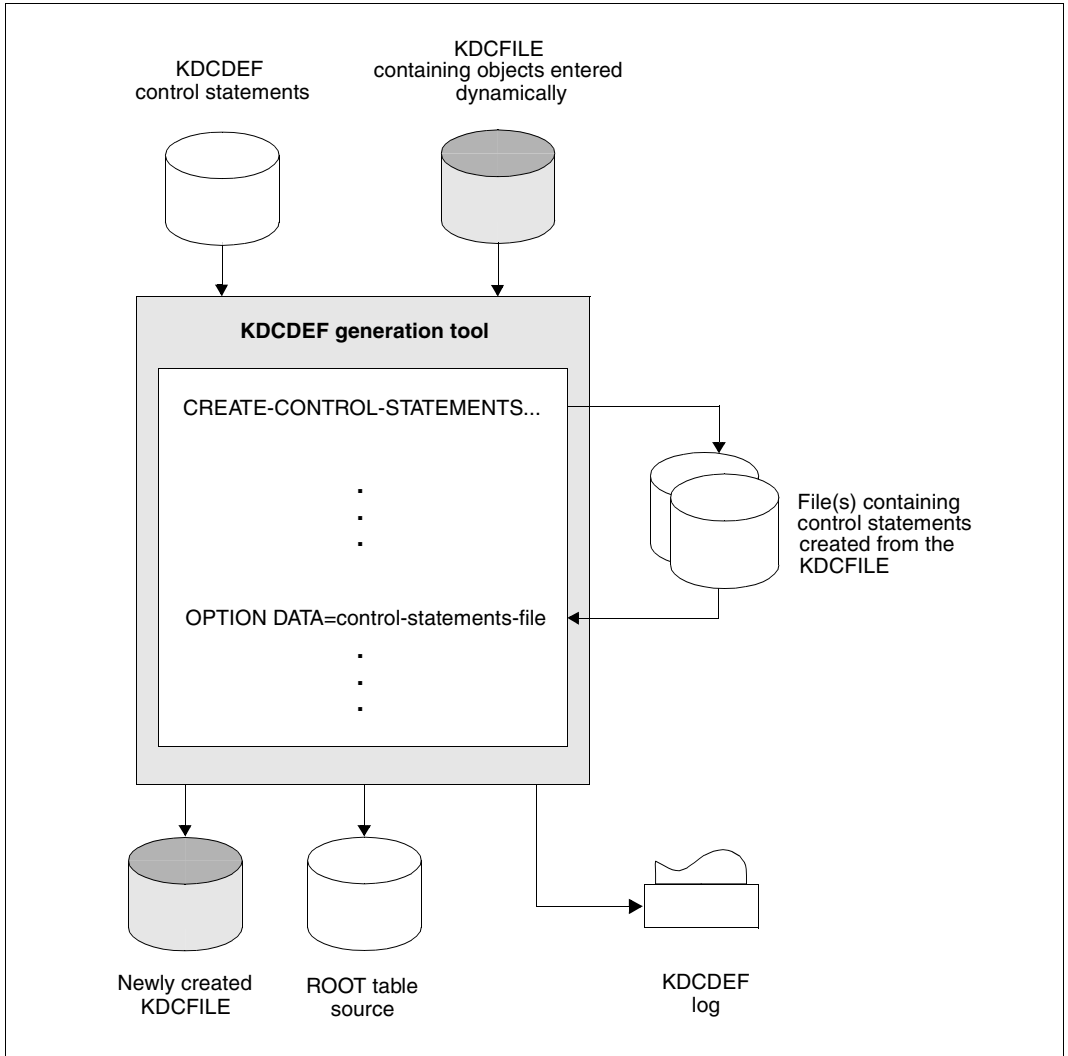


Figure 3: KDCDEF run with inverse KDCDEF

2.3 The KDCFILE

The KDCFILE contains all data required to run a UTM application. It is shared by all application processes during runtime. In its most basic form, the KDCFILE consists of a single file (a PAM file under BS2000/OSD). The KDCFILE can also be distributed over several files. For security reasons, it can be duplicated.

The KDCFILE is logically divided into three areas:

- **Administrative data**, see [page 49](#)
- **Page pool**, see [page 49](#)
- **Restart area**, see [page 52](#)

KDCDEF generation

The KDCFILE is generated during the KDCDEF run by specifying

```
OPTION... ,GEN=KDCFILE or GEN=ALL
```

in the KDCDEF statement.

The following characteristics of the KDCFILE must be specified at the KDCDEF generation:

- **Data block size**
Each area within the KDCFILE is organized in units of either 2 K or 4 K. These units are known as UTM pages. You can define the block size of a UTM page using the following control statement:

```
MAX... ,BLKSIZE={ 2K | 4K }
```

Whether a block size of 2K or 4K is to be favored depends on the sizes of the data areas (GSSB, LSSB, etc.) and the lengths of the messages that your program uses. See also [section “Page pool” on page 49](#) for more information.

- **Base name of the KDCFILE**
You specify the base name (called the *filebase* in the following) and single or dual-file operation of the KDCFILE with:

```
MAX... , KDCFILE={ filebase [, SINGLE | DOUBLE ] }
```

In the case of dual-file operation, the contents of both KDCFILE files are always identical. If one of the files is corrupted, it can be restored by simply copying the other file.

The name specified in *filebase* is also the base name of additional files and file generations of the application (for example, the system and user log file). *filebase* is therefore the base name of the application.

The significance of the base name *filebase* is different for each of the various platforms:

B

BS2000/OSD:

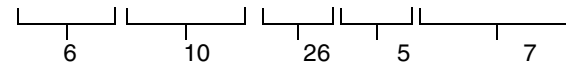
B

The full names of the files derived from *filebase* have the following format:

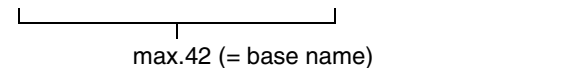
B

[:catid:] [\$userid.] prefix. suffix [(* number)]

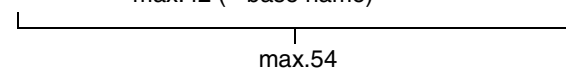
B



B



B



B

catid Catalog ID

B

userid BS2000 user ID

B

prefix Freely selectable name part

B

suffix Name part assigned by openUTM, which describes the function of the respective file

B

(*number) Identifies a file generation group, if necessary

B

B

The total length of the file name must not exceed 54 characters. The base name *filebase* can be up to 42 characters in length, including *userid* and *catid*. If no *catid* or *userid* is specified when defining the base name (*filebase=prefix*), the lengths of these fields must still be taken into consideration when determining the total length of the file name.

B

B

B

B

If file generation groups (i.e. USLOG files, SYSLOG generation group) are not used in the application, *prefix* can be up to 33 characters in length. Otherwise, *prefix* must not exceed 26 characters.

B

B

B

The KDCDEF generation tool then creates the following files:

B

– *filebase.KDCA* in the case of single-file operation (SINGLE)

B

– *filebase.KDCA* and *filebase.KDCB* in the case of dual-file operation (DOUBLE)

B

When splitting the KDCFILE, additional files are created, see [section “Splitting the KDCFILE” on page 57](#).

B

X/W *Unix systems, Windows systems:*

X/W *filebase* specifies the name of the base directory in which the KDCFILE is stored.

X/W The KDCFILE is created by KDCDEF under the *filebase* directory, where *filebase* is the fully qualified name of a directory which must be created **before** the KDCDEF run.

X/W The KDCDEF generation tool creates the following files in the *filebase* file directory:

- X/W – KDCA in the case of single-file operation (SINGLE)
- X/W – KDCA and KDCB in the case of dual-file operation (DOUBLE)

X/W In addition to the KDCFILE, the *filebase* directory generally contains the following files and subdirectories:

- X/W – files of the page pool or restart area in the case of a split KDCFILE
- X/W – directories for the user log files (USLOG files)
- X/W – the system log file (SYSLOG file) or the SYSLOG directory with the system log file generations.
- X/W – ROOT table source
- X/W – the application program
- X/W – DUMP directory: The dump files are written to this directory, as are the temporary KTA-TRCA files which are created on a PEND-ERROR and during program exchange.

- Size of the page pool

You can define the size of the page pool using the following control statement:

```
MAX...,PGPOOL=( number, warnlevel1, warnlevel2 )
```

Further information can be found in [section “Page pool” on page 49](#).

- Size of the restart area

You can define the size of the buffer and the restart area using the following control statement:

```
MAX ...,RECBUF=( number,length )
```

Further information can be found in [section “Restart area” on page 52](#).

During generation, the page pool and the restart area can be distributed over several files. Further information can be found in [section “Splitting the KDCFILE” on page 57](#).

Data security - dual-file operation of the KDCFILE

For security reasons, it may make sense to duplicate the KDCFILE (dual-file operation). If one of the files is destroyed, then you can continue working with the other KDCFILE without losing data.

Dual-file operation of the KDCFILE does not have any significant effect on I/O times (these are certainly not doubled), and therefore does not reduce performance.

With dual operation of the KDCFILE, it makes sense to store the two files on different volumes (disks). If one of the volumes is physically damaged, this ensures that a viable copy is still available.

B *BS2000/OSD:*

B You can create the files on the desired volumes by issuing appropriate `/CREATE-FILE`
B commands before the KDCDEF run or by copying the files after the KDCDEF run. When
B generating the application, you can also use the CATID parameter of the MAX statement to
B assign different CATIDs to the two files.

B A copy of the KDCFILE is maintained when you specify `MAX KDCFILE=(... ,DOUBLE)` in the
B KDCDEF generation.

X *Unix systems:*

X Under Unix systems you can store the two KDCFILES on different disks. This is generally
X only possible with the help of symbolic links (`ln -s`) to raw devices or across different file
X systems across different file systems. In this manner, you have a copy of the file even if one
X of the two disks is physically destroyed.

X A copy of the KDCFILE is maintained when you specify `MAX KDCFILE=(... ,DOUBLE)` in the
X KDCDEF generation.

W *Windows systems:*

W On Windows systems you can additionally increase data security with the operating
W resources already available. For example, you can use single-file operation for the
W KDCFILE (`MAX KDCFILE=(... ,SINGLE)`) and create a mirrored image of the disk on which
W the KDCFILE is stored on another disk. During operation, all changes to the KDCFILE are
W also made on the mirror disk. Even if one of the disks is physically destroyed, you can still
W continue working with the other hard disk without losing data.

2.3.1 Administrative data

The administrative data area contains configuration information, such as application runtime parameters, lists of all objects that can be addressed by name, administrative data on the page pool and restart area, and tables of user IDs, clients, LTERM partners, transaction codes, key and lock codes, and function keys.

All tasks and work processes of the application work with the administrative data and use the application to exchange information.

The administrative data itself is initialized using the KDCDEF generation tool. When starting the application, it is loaded into a shared memory, which can then be accessed by all tasks/work processes of the application.

B Under BS2000/OSD this memory is located in a Common Memory Pool.

X/W
X/W Under Unix systems and Windows systems the administrative data is put into a shared memory segment.

In a UTM-S application, openUTM writes the administrative data (including any modifications made in the meantime) back to the KDCFILE at certain intervals (Periodic Write). This also occurs at the end of the application run. This version of the administrative data then forms the basis for the next application run.

In a UTM-F application, openUTM writes only certain modified administrative data back to the KDCFILE, e.g. changed user passwords and generation data incorporated by means of dynamic administration.

2.3.2 Page pool

The page pool stores user data created during the application run. This includes:

- LSSBs, GSSBs, TLS blocks, and ULS blocks
- message queues, i.e. asynchronous messages (including time-driven messages) for clients, asynchronous services and service-controlled queues, and the dead letter queue, among other items
- buffered user log records (USLOG)
- service data (KB program area, last dialog message, etc.)
- dialog messages buffered after input as a result of TAC class or priority control
- output messages to clients

The active UTM application accesses the page pool via the UTM cache. During KDCDEF generation, you can define the size of the page pool (number of UTM pages) using the MAX statement:

```
MAX...,PGPOOL=( number, warnlevel1, warnlevel2 )
```

number Size of the page pool in UTM pages

warnlevel1 The first warning is output when the percentage utilization of the page pool reaches the value specified here

warnlevel2 The second warning is output when the percentage utilization of the page pool reaches the value specified here

The utility program KDCUPD provides a way of obtaining more detailed information about the source of the data stored in the page pool. It is possible to display the number of pages used for each application object, e.g. for each user. You will find more information on this subject in the [chapter “The tool KDCUPD – updating the KDCFILE” on page 577](#).

Estimating the necessary size of the page pool

Once the page pool size has been defined, it cannot be modified while the application is running. When designing a UTM application, it is therefore recommended that you estimate the page pool size which will be required during runtime. The page pool size cannot be modified until after the application has been terminated. This involves regenerating the KDCFILE using the KDCDEF generation tool, whereby the existing user data is copied then from the old KDCFILE to the new KDCFILE using the KDCUPD tool. Further information can be found in [chapter “The tool KDCUPD – updating the KDCFILE” on page 577](#).

When estimating the required page pool size, you must examine the behavior of the program units, identify which data areas are stored in the page pool, and determine their size. The following must also be noted:

- The page pool is divided into UTM pages:
a UTM page is either 2KB or 4KB in length, depending on the value of the BLKSIZE= parameter in the MAX statement.
- The following applies for the data areas GSSB, LSSB, TLS, and ULS:
each of these data areas begins on a new UTM page. Of each UTM page, 1994 bytes (in the case of a 2KB UTM page) or 4042 bytes (in the case of a 4KB UTM page) are available for user data. The remaining space is reserved by openUTM.

- The following applies to asynchronous messages:
Each message begins on a new UTM page. Of the first UTM page of a message, at least 1914 bytes (in the case of a 2KB UTM page) or 3962 bytes (in the case of a 4KB UTM page) are available for user data. Of each follow-up page occupied by the message, at least 2030 bytes (in the case of a 2KB UTM page) or 4078 bytes (in the case of a 4KB UTM page) are available for user data.

In future versions of openUTM, it is possible that less space will be provided for user data on each UTM page. When programming, therefore, you should ensure that all the available space is not exhausted.

- If an existing area is modified, openUTM stores the new area up to the end of the transaction at another location in the page pool. The area is thus temporarily duplicated.

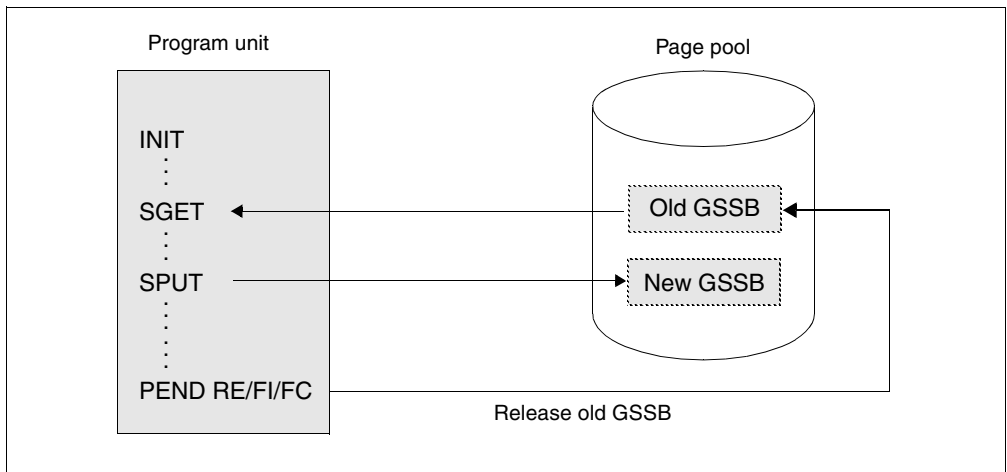


Figure 4: Dual-file operation of changed areas in the page pool



You must also allow for the volume of FPUT and LPUT messages. Make sure that the page pool is not too small.

Page pool overflow warning

It is vital that the page pool does not become full while the application is running. For this purpose, openUTM provides the following protective measures:

- There are two warning levels (percentages), which can be set during generation. If utilization of the page pool exceeds or falls short of these values, openUTM outputs UTM message K040 or K041, so that the user can respond with a MSGTAC routine.
- Local asynchronous messages and LPUT calls to write records to the USLOG file are rejected if utilization of the page pool has reached warning level 2.
- Asynchronous messages from a partner application via LU6.1 or OSI TP are rejected if utilization of the page pool has reached warning level 2. The connection is cleared. When communicating via OSI TP, the message `K119 OSI-TP error information with the insert DIA3=21` is output in the job-submitting and job-receiving application. The queued message is resent to the job-receiving application at regular intervals determined by the value in MAX CONRTIME.
- An asynchronous job issued by a terminal or TS application is reject with UTM message K101 if utilization of the page pool has reached warning level 2.

2.3.3 Restart area

KDCS calls in a program unit will result in modifications to the administrative data. openUTM collects information on all changes made within a transaction - i.e. from the first INIT call to the end of the transaction - in a process-specific storage area. Under BS2000/OSD this is a buffer in class 5 memory.

In a UTM-S application, openUTM uses the information in this buffer to create a restart data record at the end of the transaction. It then writes this data record to the restart area of the KDCFILE. The data record describes the modifications to the administrative data which were made as a result of the transaction. In the case of a warm start, it is used by openUTM to trace the effect of the transaction. The size of the restart area determines the interval at which modifications to the administrative data must be transferred to the administrative data area of the KDCFILE.

In a UTM-F application, restart data records are written only for transactions in which passwords were changed or in which administrative data was modified by means of dynamic configuration. The data records in the restart area are combined by openUTM, i.e. a UTM page in this area generally contains several data records.

During KDCDEF generation, you can define the size of the buffer and the restart area using the following statement:

```
MAX ...,RECBUF=( number,length )
```

number Size of the restart area for each process in the KDCFILE, specified in UTM pages

length Size of the buffer for each process in the main memory, specified in bytes

Setting the *length* parameter

The *length* parameter lets you reserve for each process a buffer area that is *length* bytes long in main memory. openUTM uses this area to buffer changes to administrative data while a transaction is still open and can therefore still be reset.

For *length*, you must calculate the space requirements of the application's transactions in the buffer using default values:

- In addition to the basic requirement of 40 bytes per transaction, you must also allow for the following:
 - up to 50 bytes per KDCS call, but 80 bytes per MCOM call.
On 64-bit platforms you will need to double the memory requirement.
 - up to 300 bytes per ADMI call for 32- and 64-bit platforms.
- In the case of distributed processing, the following additional requirements must be taken into consideration:
 - 300 bytes per LU6.1 communication partner
 - 200 bytes per OSI TP partner
- In the case of asynchronous administration by means of an FPUT call, please note that all FPUT NT calls from a program unit to the same administration TAC are processed by the UTM administration program in a single transaction. The individual administration commands require the following buffer space, which must be taken into consideration in *length*:
 - 0 bytes for each KDCHELP and KDCINF administration command
 - for all other administration command others
 - 300 bytes under BS2000/OSD
 - 360 bytes under Unix systems and Windows systems

B
X/W

If RECBUF=*length* is generated too small, i.e. if the buffer is not large enough for a transaction, openUTM rejects the KDCS call or reset the transaction or cancel the transaction abnormally.

Setting the *number* parameter

The *number* parameter lets you reserve for each process a buffer area whose size is *number* UTM pages in the KDCFILE. openUTM uses this area to buffer changes to the administrative data of completed transactions until the changed data is written to the KDCFILE in the next periodic write. A UTM page generally contains several data records, since these generally occupy only slightly less space than the corresponding information in the buffer.

If *number* is defined too low, KDCDEF automatically increases this to the minimum value.

When combined with the restart records, the administrative data in the KDCFILE always represents the last valid state of the application. When using UTM-S, openUTM automatically updates the administrative data in the KDCFILE (Periodic Write) before a restart area becomes full during runtime. All pages containing administrative data in which modifications have been made are written to the KDCFILE parallel to the transactions currently active. All data records written previously to the restart areas thus become obsolete.



If the restart area is large, the administrative data in the KDCFILE is updated less frequently during runtime. When performing a warm start following termination of the application, however, large quantities of data records from the restart areas must be incorporated in the KDCFILE, i.e. the warm start takes longer. The opposite applies if the restart area is small: since administrative data is updated frequently during runtime, the time required for warm start is reduced.

number should be set such that the space available in the restart area is at least a multiple of the buffer generated using the *length* parameter.

In a UTM-F application, the volume of administrative data written back to the KDCFILE is much less, e.g. changed user passwords and generation data modified by means of dynamic configuration. The *number* parameter can therefore be set lower.

2.3.4 Creating a new KDCFILE during operation

To minimize the downtime for a UTM application during a new generation, it is also possible to create a new KDCFILE for an application while the application is running. However, you must bear the following in mind:

B *BS2000/OSD:*

B The entire base name of the new KDCFILE consists of the catalog ID, user ID and prefix
B and may not be the same as the old (current) KDCFILE (the structure of the file name is
B described on [page 46](#)).

B To ensure this, use the following procedure:

B 1. In the MAX statement, enter the file name without the *userid* in the *filebase* parameter.
B And under *prefix* (see [page 46](#)) enter the same value as used for the generation of the
B "old" KDCFILE.

B 2. Start the KDCDEF run under a BS2000 user ID which is different to the one under which
B the application is running (for example, use *userid2*, if the old KDCFILE is called
B `:catid:$userid1.prefix.KDCA`).

B You can subsequently - when the application is not running - copy the KDCFILE to the
B *userid1* and execute a KDCUPD run, if necessary. You can copy the KDCFILE using:

B `/COPY-FILE FROM-FILE=$userid2.filebase.KDCA,T0-FILE=$userid1.filebase.KDCA`

B Start the KDCDEF run under *userid1* or in MAX `KDCFILE=` enter the base name with the
B user ID,. This will cause KDCDEF to interrupt the KDCDEF run with the message K404
B "DMS error D5B1 on file ...".

X/W *Unix systems, Windows systems:*

X/W You must ensure that the directory in which the new KDCFILE will be written is not the same
X/W directory as the base directory of the running UTM application.

X/W To achieve this, proceed as follows:

X/W 1. Specify the base directory *filebase* with “.” in the MAX statement, i.e. the KDCFILE will
X/W be written in the directory in which KDCDEF is started:

X/W MAX KDCFILE=(.,S) or MAX KDCFILE(.,D)

X/W 2. You start KDCDEF in a directory other than the base directory of the UTM application.

X/W You can then copy the KDCFILE to the base directory later and execute a KDCUPD run, if
X/W necessary.

X/W If you start the KDCDEF run in the base directory or specify the fully qualified base directory
X/W in MAX KDCFILE= , then KDCDEF aborts the KDCDEF run with message U185.

2.4 Performance aspects - tuning

An important factor in the performance of a UTM application is the efficiency with which openUTM can access the KDCFILE, particularly in the case of high transaction rates. With a large configuration, i.e. a large KDCFILE, it is recommended that you optimize the access times. This can be achieved in two ways:

- splitting the KDCFILE (see below)

X

- KDCFILE on raw-device (only under Unix systems, see [page 59](#))

W

- KDCFILE on stripe set (only under Windows systems, see [page 62](#))

B

In BS2000/OSD, you can also use the HIPERFILE concept to optimize performance, see the manual "BS2000/OSD - Introductory Guide to DMS".

B

2.4.1 Splitting the KDCFILE

In order to improve the I/O behavior of your application, you can split the KDCFILE by swapping out the page pool and/or the restart area of the KDCFILE. Splitting the page pool and restart area across several files is particularly advantageous if you have very high transaction rates, since openUTM then distributes the number of access operations to these areas across the various different files.

The administrative data is essentially stored in the main file KDCA. The swapped-out page pool or restart area can be split between several files by means of generation. Provided this results in the use of numerous different hardware paths, access times can be reduced considerably thereby enhancing performance.

Generation notes

You can use the following operands of the KDCDEF control statement MAX to define the areas of the KDCFILE to be swapped out during generation, and to specify the number of files created for these areas:

- Page pool files

```
MAX . . . ,PGPOOLFS=number
```

- Restart area files

```
MAX . . . ,RECBUFFS=number
```

In the case of dual-file operation, which is defined using the statement `MAX . . . ,KDCFILE=(. . . ,DOUBLE)`, these files are also maintained twice.

File names

The individual files of the KDCFILE that are contained in the swapped out areas have the same base name *filebase* as the main file KDCA and have the following names:

- Page pool files: P01A, P02A, P03A,
If you are keeping a dual KDCFILE, the files P01B, P02B, P03B, ... are also created.
- Restart area: R01A, R02A, R03A,
If you are keeping a dual KDCFILE, the files R01B, R02B, R03B, ... are also created.

Example

You want to set up your KDCFILE as follows:

- Page pool distributed across two files.
- Restart area located in a separate file.
- Duplicated file names.

For the base names, the place holder *FILEBASE* is used in this example.

X/W Under Unix systems and Windows systems, *FILEBASE* is the directory in which the files are
 X/W stored, and can be replaced, for example, by /home/userutm/base (Unix systems) or
 X/W C:\userutm\base (Windows systems).

In the KDCDEF generation you specify the following MAX statement:

```
MAX . . . , KDCFILE = ( FILEBASE , DOUBLE ) , PGP00LFS = 2 , RECBUFFS = 1 , . . .
```

KDCDEF then generates the following files:


		KDCFILE	Original	Copy
B B B B	BS2000/OSD:	Main file containing administrative data	<i>FILEBASE.KDCA</i>	<i>FILEBASE.KDCB</i>
		Page pool	<i>FILEBASE.P01A</i> <i>FILEBASE.P02A</i>	<i>FILEBASE.P01B</i> <i>FILEBASE.P02B</i>
		Restart area	<i>FILEBASE.R01A</i>	<i>FILEBASE.R01B</i>
X X X X	Unix systems:	Main file containing administrative data	<i>FILEBASE/KDCA</i>	<i>FILEBASE/KDCB</i>
		Page pool	<i>FILEBASE/P01A</i> <i>FILEBASE/P02A</i>	<i>FILEBASE/P01B</i> <i>FILEBASE/P02B</i>
		Restart area	<i>FILEBASE/R01A</i>	<i>FILEBASE/R01B</i>
W W W W W	Windows systems:	Main file containing administrative data	<i>FILEBASE\KDCA</i>	<i>FILEBASE\KDCB</i>
		Page pool	<i>FILEBASE\P01A</i> <i>FILEBASE\P02A</i>	<i>FILEBASE\P01B</i> <i>FILEBASE\P02B</i>
		Restart area	<i>FILEBASE\R01A</i>	<i>FILEBASE\R01B</i>

2.4.2 KDCFILE on raw-device (Unix systems)

X You can improve the performance of a UTM application under Unix systems considerably
 X by operating the KDCFILE on raw-device, i.e. as a character based device file. To do this,
 X create the KDCFILE on a separate disk partition, in other words a partition on which no file
 X system is stored.

X Direct access to the KDCFILE via a device file without buffering in the system kernel
 X requires less time and less resources than access via the file system when the KDCFILE is
 X stored as a normal file in the *filebase* directory. The KDCFILE is stored as a contiguous data
 X area on the disk partition. If the KDCFILE is stored as a file within a file system, then the
 X data of the KDCFILE is often stored by the system in such a way that it is distributed across
 X several storage areas which leads to increased access times.

X Splitting the KDCFILE across several files (swapping out the page pool and restart area)
 X requires you to use a **separate** disk partition for each file.

X  The raw partition of the database system used should be located on another disk.

X **Estimating the size of the required disk partition**


X To allow the system administrator to create a sufficiently large disk partition for your
X KDCFILE, you must calculate the size of the KDCFILE. This depends on the following
X factors:

- X ● the number of generated objects addressed by name
X (transaction codes, users, program units, clients and printers, key sets, remote commu-
X nication partners, connections for distributed processing, etc.)
- X ● the generated size of the page pool (see [page 49](#))
- X ● the generated size of the restart area (see [page 52](#))
- X ● the number of work processes

X To determine the size of the partition required for your KDCFILE, use KDCDEF to generate
X the KDCFILE as the file *filebase/KDCA*. Then output the size of your KDCFILE using the
X command *ls -l*. Please note that future modifications to the configuration generally affect the
X size of the KDCFILE. As a precaution, you should therefore select a larger disk partition.

X **Creating the raw special file**

X The disk partitioning is defined by the system administrator when installing the Unix system.
X Before system installation, therefore, you must inform your system administrator that you
X require disk partitions in raw-device format without file systems for your UTM applications.
X The system administrator can thus create several smaller partitions during installation,
X which can then be combined to form the storage area for your KDCFILE depending on
X requirements.

X  **CAUTION!**
X Many disks contain administrative data in the first track. This area of the disk must
X therefore not be included in the partition for the KDCFILE.

X Do **not** create a file system in the disk partition in which the KDCFILE is to be
X stored. Do **not** mount the disk partition using the *mount* command.

X Ask your system administrator to create a special file for accessing the disk partition. Make
X sure that access to the KDCFILE takes place via a character-oriented special file (raw-
X device), i.e. the name of the special file must begin with *r* and the identifier must be a *c* (first
X character output by the *ls -l* command).

X The owner of the special file must be the user ID under which the application runs. Read and
X write access to the special file must be granted exclusively to the owner (access rights 600).
X Be careful when assigning access rights to the KDCFILE, as these are the only means of
X protecting your KDCFILE against unauthorized access.

X The `ls -l` command for the special file

X `ls -l /dev/rxxxx`

X returns the following output:

X `crw----- 1 utmaw other 0,1030 Jul 14 15:13 /dev/rxxxx`

X **Writing the KDCFILE to the special file**

X There are two options for creating the KDCFILE in the disk partition.

- X ● Delete the KDCFILE that you generated when determining the file size. Using the `ln` command, create a symbolic reference between the special file and `filebase/KDCA`.
- X Regenerate the KDCFILE for your application using the KDCDEF generation tool.
- X `openUTM` writes the KDCFILE directly to the special file.

X `rm filebase/KDCA`

X `ln -s /dev/rxxxx filebase/KDCA`

X `utmpath/ex/kdcdef`

- X ● Copy the KDCFILE (generated when determining the file size) to the special file using the `cp` or `dd` command, and then delete the KDCFILE `filebase/KDCA`.
- X Using the `ln` command, create a symbolic reference between the special file and `filebase/KDCA`.

X `cp filebase/KDCA /dev/rxxxx`

X `rm filebase/KDCA`

X `ln -s /dev/rxxxx filebase/KDCA`

X In both cases, after issuing the `ln` command, use the `ls -l` command to check whether a link exists between `filebase/KDCA` and the special file:

X `ls -l /dev/rxxxx filebase/KDCA`

X Output:

X `crw----- 1 utmaw other 0,1030 Jul 14 15:13 /dev/rxxxx`

X `lrwxrwxrwx 1 utmaw other 9 Jul 14 15:49 filebase/KDCA -> /dev/rxxxx`

X Dual-file operation

X If you require dual-file operation of the KDCFILE for security reasons, you will need two disk partitions. The disk partitions should be located on different disks. Ideally, the disks should be operated by different controllers. The system administrator must create a raw special file for each KDCFILE.

X To ensure that openUTM can write each KDCFILE to the special file created for this purpose, you must create the following symbolic references:

X `ln -s /dev/rxxx1 filebase/KDCA`
X `ln -s /dev/rxxx2 filebase/KDCB`

2.4.3 KDCFILE on a stripe set in (Windows systems)

W You create the *filebase* file directory on a stripe set (Windows software RAID level 0). In a stripe set, unused areas of matching size on different hard disks are combined to form a logical drive. The data in a KDCFILE on a stripe set are therefore also distributed amongst various hard disks, resulting in faster access and therefore in higher performance for the UTM application.

W You must use stripe sets with parity (RAID Level 5) to achieve better data security. Stripe sets with parity can only be used under the Windows Server operating system.

W Please consult the Windows documentation for more information on stripe sets.

3 Notes on generating a UTM cluster application

Unlike a standalone application, a UTM cluster application is intended to be run on more than one computer. Together, these computers are known as a cluster and the individual computers on which the application is to run are known as nodes. A UTM cluster application is made up of several identically generated UTM applications (the node applications) that run on the individual nodes.

The configuration of the application, including the KDCFILES for all nodes, is created in a single generation run and is therefore always the same.

In BS2000/OSD, a UTM cluster application can be distributed across up to 16 nodes and on Unix or Windows systems across up to 32 nodes.

The computers that belong to a cluster must be equivalent in terms of hardware status and software configuration. Discrepancies involving compatible correction statuses and updates are possible. Mixed configurations, such as BS2000 and Unix computers in combination are not possible.



CAUTION!

The nodes of a cluster must always have the same system time.



You can find detailed information on operation and in particular on generating applications for UTM cluster applications in the following manuals:

- openUTM manual “Using openUTM Applications under BS2000/OSD”
- openUTM manual “Messages, Debugging and Diagnostics under Unix systems and Windows systems”

B
X/W
X/W

3.1 Generating a UTM cluster application

The generation of a UTM cluster application differs in the following ways from that of a standalone UTM application:

- There are the additional statements CLUSTER and CLUSTER-NODE as well as the operand value GEN=CLUSTER ,, in the OPTION statement, see [section “KDCDEF statements” on page 69](#).
- When a UTM cluster application is generated, UTM cluster files are also generated, see below.
- Only one copy of the KDCFILE is permitted, i.e. KDCFILE=(...,SINGLE) must be specified in the MAX statement (default value).

Please also note the following important differences that apply during a UTM cluster application run:

- In UTM cluster applications, the user data that applies globally throughout the cluster is stored in GSSB and ULS areas. In the case of UTM-F, the service data is also stored in these areas.
- The KDCFILES of the node applications contain only the user data that is local to the node.

3.1.1 UTM cluster files

A UTM cluster application is generated in a generation run during which the KDCDEF utility creates the following files:

- the cluster configuration file
- the cluster user file
- the cluster page pool files
- the cluster GSSB file
- the cluster ULS file
- an initial KDCFILE
- and the root source

The initial KDCFILE must be copied for each node application after the generation run.

The UTM cluster files generated by KDCDEF do not have to be generated as often for a UTM cluster application as the KDCFILE or the root source.

Subsequent generation runs can be performed in order to

- modify the KDCFILE and/or the root source,
- regenerate the UTM cluster files. You can perform a KDCUPD run for the UTM cluster application in order to take over the data from the previous UTM cluster files into the newly generated files, see [section “Updating the KDCFILE and UTM cluster files for UTM cluster applications” on page 590](#).

If changes are made to the configuration, a new initial KDCFILE can, for instance, be created with additional objects in a subsequent generation run.

The initial KDCFILE must be copied for each node application after the generation run.



It must be possible to access the UTM cluster files and the KDCFILES of all node applications from all node applications. See also the section "UTM cluster application under BS2000/OSD" in the openUTM manual "Messages, Debugging and Diagnostics under Unix systems and Windows systems" and the section "UTM cluster application under Unix Systems" in the openUTM manual "Using openUTM Applications under Unix systems and Windows systems".

[Figure 5](#) shows what files are created when you define a UTM cluster application.

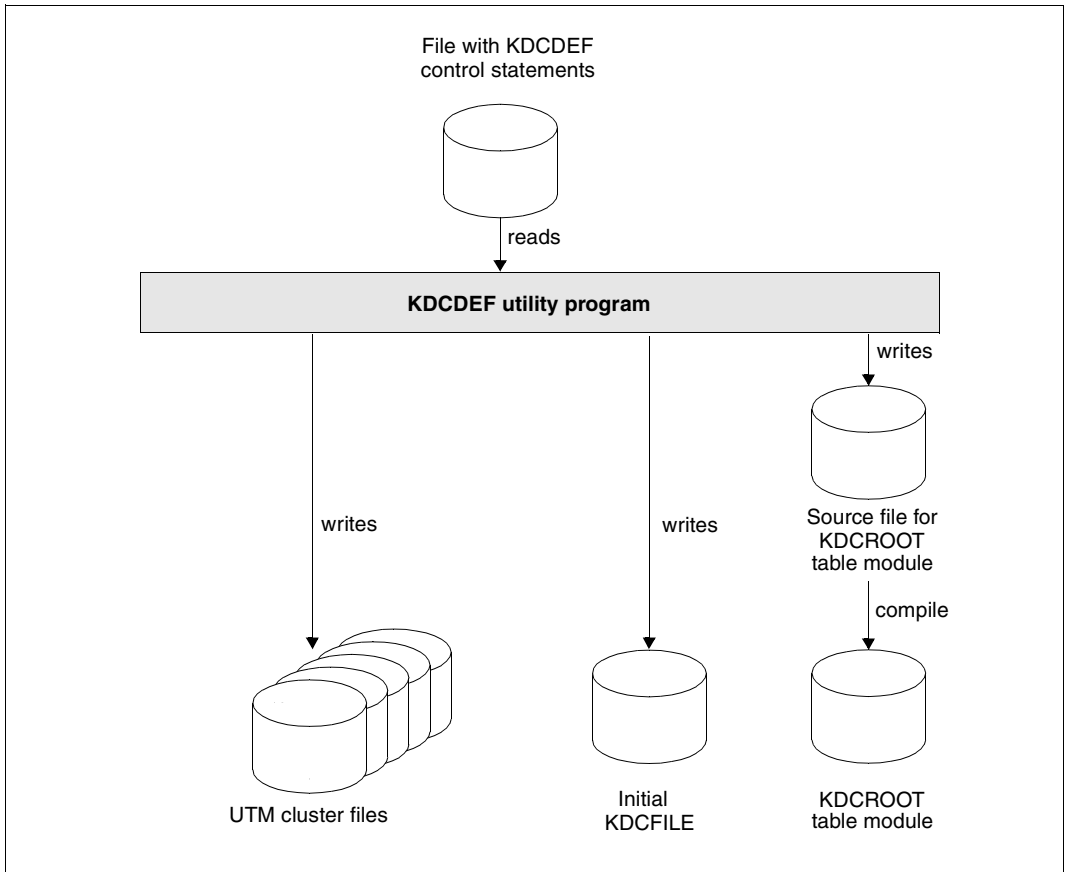


Figure 5: The result of the KDCDEF run (with `OPTION ...,GEN=(KDCFILE, ROOTSRC, CLUSTER)`) for a UTM cluster application.

If you also specify `GEN=CLUSTER` with the `OPTION` statement, a cluster configuration file is created together with the following files.

- Cluster user file for managing user IDs in a UTM cluster application.
- Cluster page pool files for storing user data in a UTM cluster application and for managing the cluster page pool.
- Cluster GSSB file and cluster ULS file for managing GSSB and ULS in a UTM cluster application.

If you specify `OPTION GEN=CLUSTER`, then you must also specify a `CLUSTER` statement and at least two `CLUSTER-NODE` statements.

Cluster configuration file

The cluster configuration file contains information on all node applications of a UTM cluster application and specifications on data that is global to the cluster. It is used jointly by all node applications of a UTM cluster application. The cluster configuration file is generally only created once for a UTM cluster application. You can only use a new cluster configuration file after all the node applications of a UTM cluster application have been terminated.

B KDCDEF creates the cluster configuration file under BS2000/OSD with the file name *cluster_filebase*.UTM-C.CFG. *cluster_filebase* is the name specified under CLUSTER-FILEBASE in the CLUSTER statement. The cluster configuration file can be renamed in BS2000/OSD, but the name suffix .UTM-C.CFG must be retained.

X/W On Unix systems and Windows systems, the cluster configuration file is created with the file name UTM-C.CFG in the directory specified by *cluster_filebase*. The cluster configuration file can be copied to a different location to operate the UTM cluster application.

Cluster user file

The cluster user file is used for managing users in a UTM cluster application.

The cluster user file is generally only created once for a UTM cluster application. The file can be extended during operation of a UTM cluster application. This always happens when the administrator defines new users for a UTM cluster application. You must therefore always also specify the cluster user file during subsequent generation runs for creating a new KDCFILE. The entries in the new KDCFILE are merged with the entries in the existing cluster user file and where necessary, KDCDEF extends the cluster user file to include entries for new users.

B KDCDEF creates the cluster user file under BS2000/OSD with the file name *cluster_filebase*.UTM-C.USER. *cluster_filebase* is the name specified under CLUSTER-FILEBASE in the CLUSTER statement. The cluster user file can be renamed in BS2000/OSD, but the name suffix .UTM-C.USER must be retained.

X/W On Unix systems and Windows systems, the cluster user file is created with the file name UTM-C.USER in the directory specified by *cluster_filebase*. The cluster user file can be copied to a different location to operate the UTM cluster application.

Cluster page pool files

The cluster page pool files are used to record user data that is managed for the entire cluster in a UTM cluster application. This data consists of the GSSBs, ULS and the user service data. The number of cluster page pool files is defined at generation time. Between one and a maximum of ten files can be created.

B In BS2000/OSD, KDCDEF creates the cluster page pool files with the file names
B *cluster_filebase*.UTM-C.CPnn, nn= 01, 02 up to a maximum of 10. Here, *cluster_filebase* is the
B name specified for CLUSTER-FILEBASE in the CLUSTER statement.

X/W On Unix systems and Windows systems, the cluster page pool files are stored with the file
X/W name UTM-C.CPnn in the directory that was defined using *cluster_filebase*, nn= 01, 02 up to
X/W a maximum of 10.

A control file for the cluster page pool is also always created. The name of this file includes the specification UTM-C.CPMD.

Cluster GSSB file

The cluster GSSB file is used for managing GSSBs in a cluster application.

B In BS2000/OSD, KDCDEF creates the GSSB file with the file name *cluster_filebase*.UTM-
B C.GSSB where *cluster_filebase* is the name specified under CLUSTER-FILEBASE in the
B CLUSTER statement.

X/W In Unix systems and Windows systems, the cluster GSSB file is created with the file name
X/W UTM-C.GSSB in the directory that was defined using *cluster_filebase*.

The cluster GSSB file is generally only created once for a cluster application. The file can be extended while an application is running. This is done whenever the space left in the file is no longer sufficient to accept the current management information.

Cluster ULS file

The cluster ULS file is used for managing ULSs in a cluster application.

B In BS2000/OSD, KDCDEF creates the cluster ULS file with the file name
B *cluster_filebase*.UTM-C.ULS where *cluster_filebase* is the name specified under CLUSTER-
B FILEBASE in the CLUSTER statement.

X/W In Unix systems and Windows systems, the cluster ULS file is created with the file name
X/W UTM-C.ULS in the directory that was defined using *cluster_filebase*.

The cluster ULS file is generally only created once for a cluster application. The file can be extended while an application is running. This is done whenever the space left in the file is no longer sufficient to accept the current management information.

3.1.2 KDCDEF statements

Special generation statements are required for generating a UTM cluster application:

- You define global properties of a UTM cluster application using the CLUSTER statement. See [page 299](#). These include, for instance
 - the cluster filebase
 - the BCAMAPPL name for cluster-internal communication
 - timers for monitoring
 - a failure command and an emergency command to be called if a node fails
 - specifications on the cluster page pool files (number, warning level, size)
 - specifications concerning behavior on user sign-on as well as on deadlock handling.
- You specify node-specific properties for each node application with the CLUSTER-NODE statement. See [page 310](#). These include, for instance
 - the base name of the KDCFILE, the user log file and the system log file SYSLOG
 - the host name of the node

You must issue a separate CLUSTER-NODE statement for each node application.



- If specifications in the CLUSTER statement or CLUSTER-NODE statements are modified then it is always necessary to create a complete, new generation. This means that the KDCFILE and the cluster files must be regenerated. The only exception are dynamic increases in the size of the cluster page pool.
- If the UTM cluster files are to be created on generation, you must specify the GEN=CLUSTER parameter in the OPTION statement (see also [page 417](#)).
- You must specify MAX BLKSIZE=4K during KDCDEF generation for UTM cluster applications. This value is implicitly generated for applications on 64-bit Unix systems.
- It is not possible to generate a UTM cluster application with two copies of the KDCFILE, i.e. the value MAX KDCFILE=(..., SINGLE) must be specified (this is the default value).

3.1.3 Initial KDCFILE

In the same way as with a standalone application, the initial KDCFILE is stored under the base name that you specify in the KDCFILE operand of the MAX statement.

Each node application uses a copy of the initial KDCFILE at runtime of a node application. To allow this, you must copy the initial KDCFILE once for each node application after the generation run.

Because each node application is monitored by another node application, all node applications must have mutual access to all KDCFILES.



You will find information on starting and monitoring the node applications and detecting failures in the following manuals:

- openUTM manual “Using openUTM Applications under BS2000/OSD”
- openUTM manual “Using openUTM Applications under Unix systems and Windows systems”

B
X/W
X/W

3.2 Generating a reserve node application

During generation with KDCDEF, you have the option of creating reserve node applications with provisional values. You can subsequently use the administration facilities to change the host name and the base name of the KDCFILE of these node applications. The node application must not be active when this is done.

- ▶ To do this, specify the provisional, node-specific base name of the KDCFILE and the host name of the reserve node using the CLUSTER-NODE statement. See [page 310](#).
- ▶ At a subsequent time, you use KC_MODIFY_OBJECT administration statement to change the node-specific properties of the reserve node application: Specify the object type KC_CLUSTER_NODE to assign the spare node application actual values for the host name of the cluster node and the base name of the KDCFILE of the node application.



For further details on possibilities for using reserve node applications, refer to the openUTM manual “Using openUTM Applications under BS2000/OSD” or the openUTM manual “Using openUTM Applications under Unix systems and Windows systems”.

For detailed information on changing the node-specific properties using the administration facilities, refer to openUTM manual “Administering Applications”.

3.3 Using global memory areas

GSSB and ULS

In openUTM as of V6.1, the use of the UTM memory areas GSSB and ULS is possible in UTM cluster applications. These memory areas can therefore be read and written globally throughout the cluster. The user data is stored in cluster page pool files (see [page 68](#)) and the management data for the GSSB and ULS areas is stored in the cluster GSSB file and cluster ULS file, respectively, see [page 68](#).

You can use the KDCDEF statement `CLUSTER ... DEADLOCK-PREVENTION` to specify whether or not openUTM is to perform additional checks to prevent deadlocks if memory areas are locked.

GSSB and ULS data are also saved in the case of UTM-F.

TLS

The UTM TLS memory areas are created locally to the nodes in UTM cluster applications, as each TLS is assigned to an LTERM or (OSI-)LPAP and a connection can be established to every LTERM or (OSI-)LPAP in every cluster node at any time. A separate version of the memory area therefore exists in each cluster node.

3.4 Using users with `RESTART=YES`

In UTM cluster applications, a service restart is possible for all genuine user IDs that have been generated with `USER ..., RESTART=YES`.

The open services of such users can be continued in any node application provided that the open service is not bound to a node application, see below. A bound service can only be continued in the node application to which it is bound.



Note on UTM-F

The following data is lost when a node application is terminated:

- The service data of services which have a job receiver in a distributed transaction
- The service data of inserted services

Furthermore, service data is not saved every time a transaction is terminated but instead only when a user signs off. This means that if an application is terminated abnormally then the service data of users who were signed on at the node application at the time of the abnormal termination is lost.

Service that is bound to a node application

An open service is bound to a node application if the service

- has a job-receiver service
- or has started a SESAM transaction
- or is an inserted service resulting from service stacking.

In addition, following abnormal termination, an open service is bound to a node application if the user was signed on at the node application at the time the application was terminated.

If a user who wants to sign on at another node application even though his service is bound to a node application then the sign-on attempt is rejected if


- the node application to which the service is bound is running,
- or the bound service has a transaction in the state PTC (prepare to commit),
- or the UTM cluster application was generated with `CLUSTER ... ABORT-BOUND-SERVICE = NO`.

Connection user IDs

The service restart for connection user IDs is bound to the connection and therefore to the node application. A connection user ID generated with `RESTART=YES` can have a service context that permits restarts in every node application.

3.5 Special issues in BS2000/OSD

B If the default catalog of the user ID under which the UTM cluster application is to be started,
 B does not correspond to the CATID of the shared pubset on which the KDCFILE is to be
 B stored, you must specify the CATID in the generation.

B  **CLUSTER-NODE statement** on [page 310](#)
 B The CATID operand is used to specify the CATIDs of the global cluster files.

3.6 Special issues on Unix systems and Windows systems

- X/W You must create the appropriate directories for storing the global cluster files before the generation run. These must exist and be accessible before the generation run.
- X/W The name for MAX KDCFILE and CLUSTER-NODE FILEBASE must not exceed 27 characters for UTM cluster applications.
- X/W The value you enter for MAX KEYVALUE must not exceed 3900 for UTM cluster applications on 64-bit Unix systems.

3.7 Special issues with LU6.1 connections

More sessions (LSES statements) than connections (CON statements) can be assigned to an LPAP partner for a UTM cluster application. KDCDEF warns you of this with message K438, but a KDCFILE is created.



For information on what you need to consider for LU6.1 communication between a standalone application and a UTM cluster application, refer to the [section “LU6.1-LPAP bundles of a standalone application with a UTM cluster application” on page 90](#).

4 Generating applications for distributed processing

This chapter provides a summary of the most important generation notes for applications with distributed processing and describes how the UTM generation is coordinated with the generation of the transport system.

The term distributed processing is used to describe server-server communication using the LU6.1 and OSI TP protocols. These protocols are used to implement global transaction processing. The OSI TP protocol also makes it possible to communicate with OpenCPIC clients and LU6.2 applications. The generation of OpenCPIC clients is described in [section “Connecting clients to the application” on page 128f](#). More information about connecting to a LU6.2 application can be found in the openUTM manual “Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications”.

The basic principles of distributed processing are introduced in the openUTM manual “Concepts und Functions”.

To generate applications with distributed processing, you must first ensure that the individual applications have been generated without errors, and that the generation data of all applications involved has been coordinated. Since the KDCDEF generation tool can only check the generation data of a single application for consistency and syntactic accuracy, conflicts generally cannot be identified until the applications begin to interact with each other, e.g. during connection setup.

If you use WinAdmin to generate distributed UTM applications, WinAdmin checks the consistency of the cross-application generation data. The vast majority of generation errors can be avoided in this manner.

Generation when standalone UTM applications are to be linked to UTM cluster applications



You will find notes on generation when standalone UTM applications are to be linked to UTM cluster applications in the [section “LU6.1-LPAP bundles of a standalone application with a UTM cluster application” on page 90](#) and in the [section “OSI-LPAP bundles” on page 105](#).

4.1 Distributed processing via the LU6.1 protocol

Before discussing the rules and recommendations for generating UTM applications with distributed processing, a number of SNA terms relevant for configuration are explained below in context.

SNA terms are shown in *italics* in the next section. More information on SNA terms can be found in the openUTM manual “Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications”.

4.1.1 Transport connections and SNA sessions

Communication between two applications is, from the openUTM point of view, carried out using transport connections (in the sense of TRANSDATA), via which the SNA sessions are handled.

The sessions are identified using *session names*. The session names serve to restart interrupted communication between two applications. If, prior to an interruption, communication via one of the possible transport connections is taking place using a given session name, then when the session is restarted, it is started under the same session name, but not necessarily using the same transport connection.

A session is defined using the KDCDEF control statement LSES, while its characteristics (e.g. the way in which it is opened, controlled, and managed) are defined using the SESCHA control statement.

The session name can be likened to the USER name in UTM applications: a USER can also continue an interrupted service on another terminal, thus using a different transport connection. In order to ensure that the session name in two connected applications does not have to be the same, the session name is made up of two parts (symbolized using the '+' character):

session-name = local-session-name+remote-session-name.

Each part of the session name is a maximum of 8 characters in length, thus the entire session name has maximum length of 16 bytes. The *local-session-name* refers to a common session in the local application, and the *remote-session-name* refers to the same session in the remote application. This means that the both the local and remote applications are required to know the session names of the partner application. The *local-session-name* uses the USER name defined in the local application to create a common name for the local application, and the *remote-sessionname* uses the USER names defined in the remote application to create a common name for the remote application.

At the start of a service, the "user ID" field in the KB header contains a local session name if the requester is a remote LU6.1 application.

A session is exclusively occupied for the duration of the dialog between the job-sending application and the job-receiving application (bracketing). In other words, another job-sending application will be required to:

- wait until the session has been released or
- repeat its job later, as it will be rejected at this time or
- occupy a different free session and start its job from there. The prerequisite for this is that there are several transport connections and sessions available for the remote application.

The opening and closing of a session is always controlled by one of the partner applications. This application is then referred to as the *primary logical unit* or *PLU*. However, the initiative for opening a session may come from **both** applications involved.

When opening a session, the partners agree on which application is to be responsible for managing the brackets, i.e. for controlling the reservation of the session by jobs. This application is known as the *contention winner*, while the other application is referred to as the *contention loser*. In order to submit a job to the partner, the contention winner can reserve a session without consulting the contention loser beforehand. The contention loser, on the other hand, must request a session from the contention winner.

4.1.2 Generation notes

When generating UTM applications that are to communicate using the LU6.1 protocol, you must bear the following information in mind.

1. In each application, either one or two LPAP statements and the appropriate SESCHA, CON and LSES statements must be generated for each of the partner applications.

Only one LPAP statement is required for a partner application if only one of the two applications is to be sending jobs. However, if both applications are intended to send jobs to the partner application, then both applications will require two LPAP statements.

2. An LPAP that is used mainly to send jobs is generated in its SESCHA statement using CONTWIN=NO; this ensures that the local application becomes the contention winner for this LPAP. The corresponding LPAP in the partner application must then be generated with CONTWIN=YES.
3. The same number of CON and LSES statements must be generated for each LPAP; the number of CON or LSES statements determines the number of parallel connections to the partner application that are possible via this LPAP.
4. For each connection/each session, one CON or one LSES statement must be generated in each of the partner applications.

For each CON statement, the CON name and the BCAMAPPL name in the one application must correspond to the names in the partner application.

In the same way, for each LSES statement, the LSES name and the RSES name of the one application must correspond to those in the partner application.

5. All CON and LSES statements of an LPAP must address the same partner application and must also be assigned to a single LPAP name in the partner application. It is not permitted to generate several CON statements leading to different applications for one LPAP name.

It is also not permitted to generate several CON statements for a single LPAP which are assigned to different LPAP statements via the corresponding CON statements in the partner application.

This generation error cannot be recognized by openUTM, but will lead to errors during connection or session establishment and during session restart.

6. In order to establish several parallel connections between two applications, a UTM application opens several transport system end points at the transport system. Each transport system end point of a UTM application is generated using its own BCAMAPPL statement.

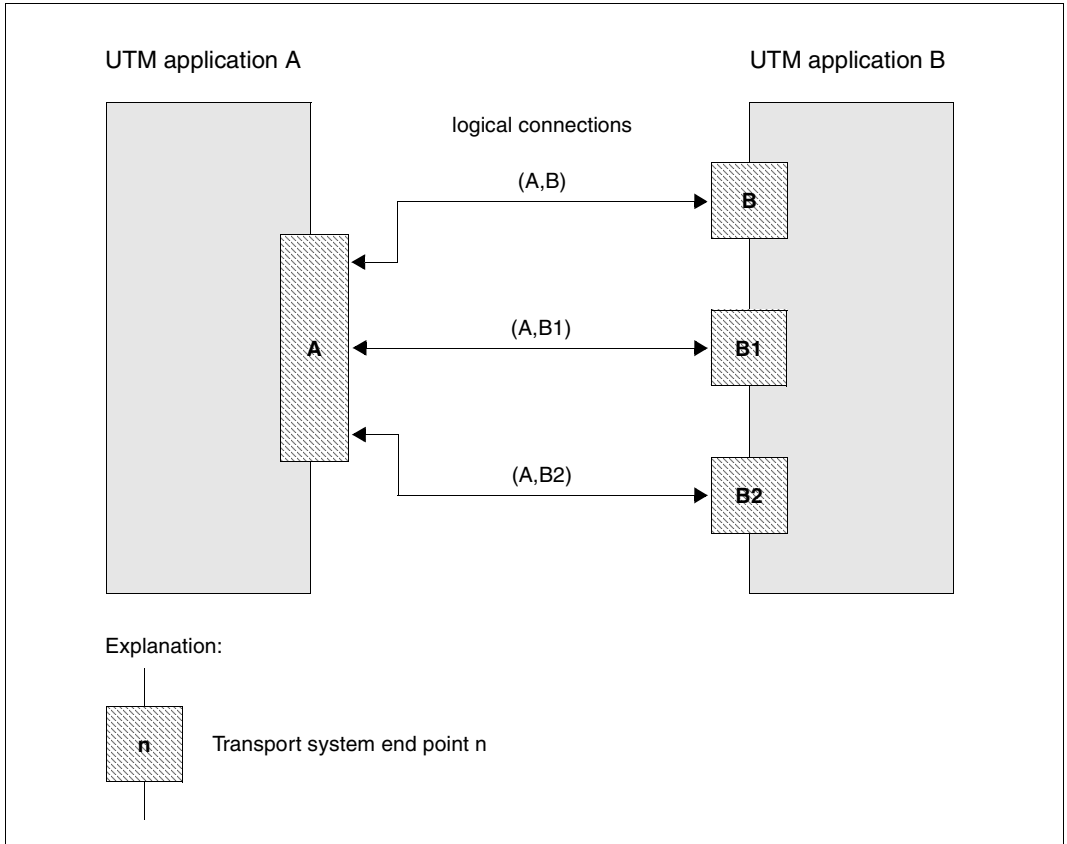


Figure 6: Two applications with several transport connections

In the example above, A and B are the names of the applications as specified using the MAX APPLNAME= statement; B1 and B2 are defined using separate BCAMAPPL statements.

Terminals are able to establish connections to application A using the application name A and to application B using the application name B. But application A is able to connect to application B using the application names B, B1 or B2.

- B From the network administration point of view, the UTM application B consists of several
- B BCAM applications. BCAM administration commands for one of the application names have
- B an effect on the entire UTM application B. So in other words, a /BCLOSE B command not
- B only terminates UTM application B, but also signs the applications B1 and B2 off from
- B BCAM.

Between two given transport system end points of both applications only a single transport connection can be established. If two transport system end points are generated in one of the applications and three in the other, then up to six parallel connections can be established between the two applications.

If both a contention winner LPAP and a contention loser LPAP are generated for a partner application (SESCHA statement), then transport system end points (BCAMAPPL statement) are established via the contention winner connections and should not be used simultaneously for the contention loser connections! This means that both contention winner and contention loser LPAPs are generated in a single application, thus the BCAMAPPLs of this application should be split into two disjunctive groups, where the BCAMAPPLs of the one group are assigned only to contention winner connections and the BCAMAPPLs of the other group are only ever used for contention user connections.

4.1.3 Procedure when generating LU6.1 connections

When generating two applications that are to communicate using the LU6.1 protocol, you should proceed as described below.

1. LPAP and SESCHA statements

First you must decide whether the two applications are to be sending jobs to each other on an equally regular basis, or whether one of the applications is to be sending jobs more frequently than the other.

In the first scenario, both applications must be generated with two LPAP statements each; in the second scenario the applications require one LPAP statement each. In this case, that LPAP statement that is designed to send more jobs than it receives is generated with SESCHA ...,CONTWIN=NO; the corresponding LPAP in the partner application is generated with SESCHA ...,CONTWIN=YES. When you have two LPAP statements in an application, one should be generated with SESCHA ...,CONTWIN=NO and the other with SESCHA ...,CONTWIN=YES.



LPAP statement on [page 344](#)

The following operands can be used to define an LPAP partner as the logical connection point for the partner application.

- *lpapname*
LPAP partner name; this is the logical name of the partner application via which the program units of the local application and the partner application communicate. *lpapname* is only significant in the local application.
- SESCHA=
The session characteristics for communication between local application and partner application as defined under *sescha_name* in the SESCHA statement are assigned to the LPAP partner.
- PERMIT=
Specifies the level of authorization (right to carry out administration and preselection functions) of the partner application.
- QLEV=
Specifies the maximum number of asynchronous messages that are permitted to wait in the Message Queue of the LPAP partner.
- STATUS=
This defines whether the partner application is able to work with the local application immediately the local application is started, or whether the administrator must first set the status to ON.

- **BUNDLE=**

Makes the LPAP a slave LPAP of an LU6.1-LPAP bundle and specifies the associated master LPAP.



- **SESCHA statement** on [page 470](#)

You can use the following operands to define the session characteristics that are to be assigned to one of the LPAP partners and therefore to the partner application that connects via this LPAP partner.

- *sescha_name*

Defines the name under which the session characteristics are collected. This name is entered in the LPAP statement in the operand SESCHA= to assign the session characteristics to a LPAP partner.

- **CONTWIN=**

Specifies whether the local application is the contention winner (NO) or contention loser (YES). The contention winner application manages the session and controls the utilization of the session by jobs.

Default: If PLU=N, the local application is the contention loser, otherwise it is the contention winner.

- **PLU=**

Specifies which application is able to initiate the session, or in other words, whether the partner application is the primary logical unit (PLU) (YES) or the local application (NO).

PLU=Y must be specified for one of the participating applications, and PLU=N for the other.

- **CONNECT=**

Specifies whether the local application is to connect automatically to the partner application on application startup (YES) or whether the connection to the partner application is to be carried out by means of an administration command (NO).

Example

Application A sends jobs to Application B via LPAP B1 and application B sends jobs to application A via LPAP A2.

Application A:

LPAP B1, SESCHA=B1
SESCHA B1, CONTWIN=NO, PLU=YES

LPAP B2, SESCHA=B2
SESCHA B2, CONTWIN=YES, PLU=NO

Application B:

LPAP A1, SESCHA=A1
SESCHA A1, CONTWIN=YES, PLU=NO

LPAP A2, SESCHA=A2
SESCHA A2, CONTWIN=NO, PLU=YES

2. BCAMAPPL statements

The next thing to do is to specify how many parallel connections are to be generated between two LPAPs. In accordance with the number you specify, the BCAMAPPL statement is used to generate additional transport system endpoints for both applications. Only one connection may be established between each transport system endpoint and the transport system endpoint of the partner application. Should, for example, nine parallel connections be generated between two LPAPs, then at least three BCAMAPPL statements will be required on each side.

If two LPAPs to the partner application are generated in an application, the BCAMAPPLs of this application should be split into two disjunctive groups. The first LPAP communicates using just the BCAMAPPLs of the first group, and the second LPAP uses the BCAMAPPLs of the second group.

**BCAMAPPL statement** on [page 291](#)

The following operands are used to define an additional application name for parallel connections to the communication partner.

- *appliname*
Additional (BCAM) name of the UTM application.
- T-PROT
Specifies the transport protocol. NEA is the default setting.

If an application communicates with several partner applications, the BCAMAPPLs used to communicate with the one application may also be used to communicate with the other applications.

3. CON and LSES statements

It is then necessary to assign one CON and one LSES statement to each LPAP statement for each parallel connection via this LPAP. Each CON and each LSES statement must be generated in each of the participating applications and both of these generations must correspond to each other.

Thus:

- Each CON name in the one application corresponds to a BCAMAPPL name in the other application and vice versa.
- Each LSES name in the one application corresponds to an RSES name in the other application and vice versa.



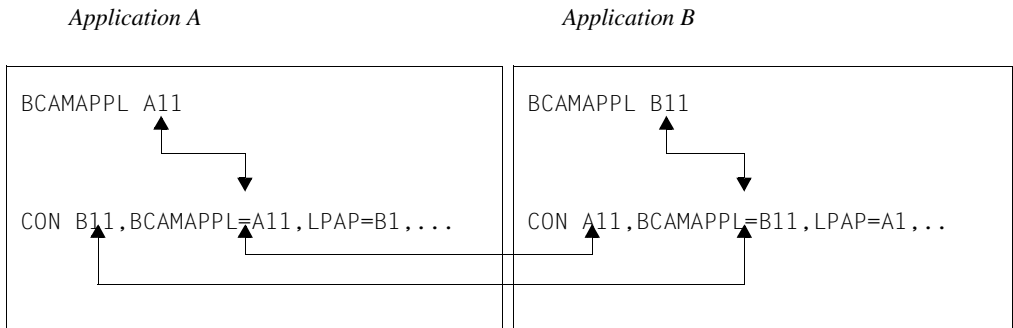
CON statement on [page 313](#)

The following operands can be used to assign the LPAP partner in the local application to the real partner application.

- *remote_applname*
Name of the partner application with which communication is to take place via the logical connection.
- BCAMAPPL=
Refers to a name of the local application as specified in the control statement MAX or BCAMAPPL. You cannot specify a BCAMAPPL name for which a T-PROT=SOCKET has been generated.
- LPAP=
Name of the LPAP partner application to which the connection is to be established. The name of the LPAP partner via which the partner application connects must be defined using the statement LPAP *lpapname*.
Specifying several CON statements with the same *lpapname* allows you to generate parallel connections to the partner application.
- PRONAM=
Name of the partner computer.

The CON statements that are used to describe the connection to the partner application in the local application and the connection to the local application in the partner application refer to the **same** connection. CON statements must therefore always be entered in pairs. When using parallel sessions, several CON statements are generated for an LPAP partner.

In the example, the assignment is made between the LPAP partner B1 (as generated in application A) and the LPAP partner A1 (as generated in application B):



LSES statement on [page 347](#)

The following operands can be used to agree the same session names for the connection and assign them to the LPAP partner.

- *local_sessionname*
Name of the session in the local application.
- RSES=
Name of the session in the partner application.
- LPAP=
Name of the LPAP partner that is assigned to the partner application.
local_sessionname is used for communication with the partner application that is assigned to the LPAP partner *lpapname* in the local application.

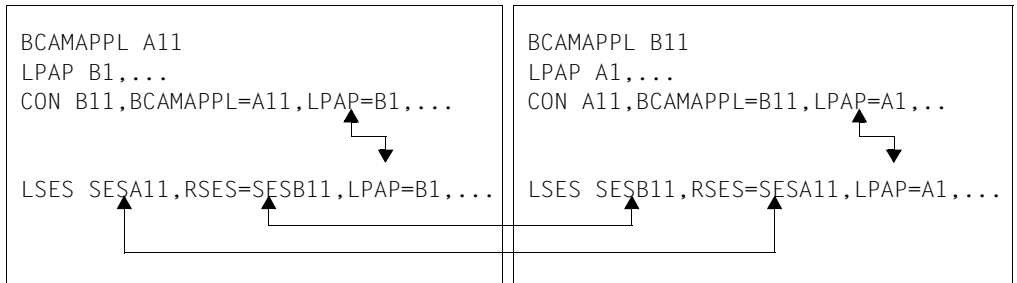
Session names are agreed in the local application and partner application. LSES statements must therefore always be entered in pairs. Since the session name is assigned to the LPAP partners, the LPAP partner assignment defined in the LSES statement must be identical to that defined in the CON statements.

If two LPAP partners are assigned to each other, the LSES and RSES names agreed in the LSES statements must match (see example below). In the case of parallel sessions, several LSES statements are entered with different session names for an LPAP partner *lpapname*.

The previous example can now be extended as follows.

Application A

Application B



*Example***Application A:**

```
BCAMAPPL A11
BCAMAPPL A12
LPAP B1, SESCHA=B1
SESCHA B1, CONTWIN=NO, PLU=YES
```

```
CON B11, BCAMAPPL=A11, LPAP=B1
CON B12, BCAMAPPL=A11, LPAP=B1
CON B11, BCAMAPPL=A12, LPAP=B1
CON B12, BCAMAPPL=A12, LPAP=B1
LSES SESA11, RSES=SESB11, LPAP=B1
LSES SESA12, RSES=SESB12, LPAP=B1
LSES SESA13, RSES=SESB13, LPAP=B1
LSES SESA14, RSES=SESB14, LPAP=B1
```

```
BCAMAPPL A21
BCAMAPPL A22
LPAP B2, SESCHA=B2
SESCHA B2, CONTWIN=YES, PLU=NO
```

```
CON B21, BCAMAPPL=A21, LPAP=B2
CON B22, BCAMAPPL=A21, LPAP=B2
CON B21, BCAMAPPL=A22, LPAP=B2
CON B22, BCAMAPPL=A22, LPAP=B2
LSES SESA21, RSES=SESB21, LPAP=B2
LSES SESA22, RSES=SESB22, LPAP=B2
LSES SESA23, RSES=SESB23, LPAP=B2
LSES SESA24, RSES=SESB24, LPAP=B2
```

Application B:

```
BCAMAPPL B11
BCAMAPPL B12
LPAP A1, SESCHA=A1
SESCHA A1, CONTWIN=YES, PLU=NO
```

```
CON A11, BCAMAPPL=B11, LPAP=A1
CON A11, BCAMAPPL=B12, LPAP=A1
CON A12, BCAMAPPL=B11, LPAP=A1
CON A12, BCAMAPPL=B12, LPAP=A1
LSES SESB11, RSES=SESA11, LPAP=A1
LSES SESB12, RSES=SESA12, LPAP=A1
LSES SESB13, RSES=SESA13, LPAP=A1
LSES SESB14, RSES=SESA14, LPAP=A1
```

```
BCAMAPPL B21
BCAMAPPL B22
LPAP A2, SESCHA=A2
SESCHA A2, CONTWIN=NO, PLU=YES
```

```
CON A21, BCAMAPPL=B21, LPAP=A2
CON A21, BCAMAPPL=B22, LPAP=A2
CON A22, BCAMAPPL=B21, LPAP=A2
CON A22, BCAMAPPL=B22, LPAP=A2
LSES SESB21, RSES=SESA21, LPAP=A2
LSES SESB22, RSES=SESA22, LPAP=A2
LSES SESB23, RSES=SESA23, LPAP=A2
LSES SESB24, RSES=SESA24, LPAP=A2
```

Notes

- In the case of applications with distributed processing, the *length* value specified in the statement `MAX...,RECBUF=(number,length),...` may have to be increased. Further information can be found in [section “Restart area” on page 52](#).
- The behavior of an application can be influenced by the choice of timer (operand `IDLETIME=` of the `SESCHA` statement, operands `CONCTIME` and `PTCTIME` of the `UTMD` statement).

4.1.4 LU6.1-LPAP bundles

LU6.1-LPAP bundles allow messages to be distributed automatically across several LPAP partners. 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 LU6.1-LPAP bundle, openUTM is responsible for distributing the messages to the partner application instances. To achieve this, the program units in the APRO call must address the MASTER-LU61-LPAP.

One application scenario for distributing messages in this way is communication between a UTM application and a UTM cluster application. This allows messages to the UTM cluster application to be distributed across the individual node applications. You will find detailed information on this in the [section “LU6.1-LPAP bundles of a standalone application with a UTM cluster application” on page 90](#).

An LU6.1-LPAP bundle consists of a master LPAP and multiple slave LPAPs. The slave LPAPs are assigned to the master LPAP on generation. In normal circumstances, the individual slave LPAPs address different partner applications.

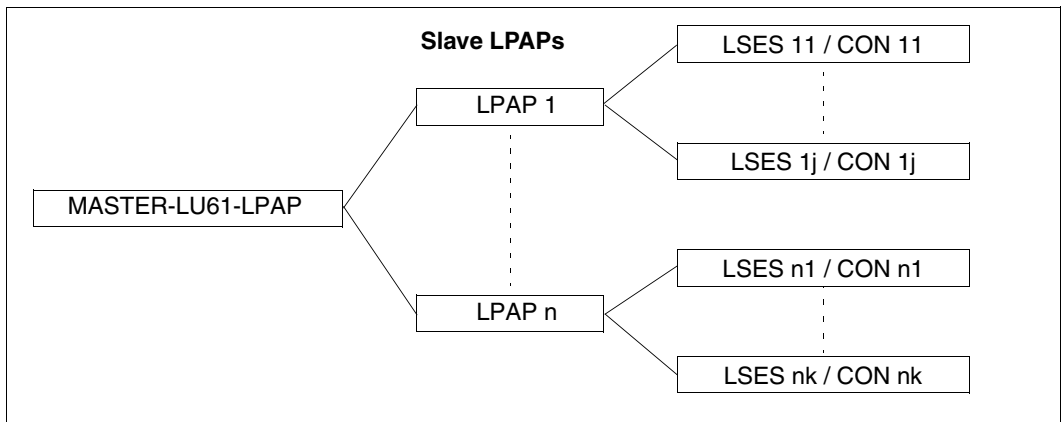


Figure 7: Example of an LU6.1-LPAP bundle

Generating an LU6.1-LPAP bundle



MASTER-LU61-LPAP statement on [page 366](#)

Specifies the name and properties of the master LPAP in an LU6.1-LPAP bundle.

- *master-lpap-name*
Name of the master LPAP.
- STATUS=
Specifies whether messages can be sent to this LPAP bundle.



LPAP statement on [page 344](#)

The following properties must be specified to generate a slave LPAP:

- *lpap-name*
Name of the slave LPAP.
- BUNDLE=master-lpap-name
Name of the master LPAP. The master LPAP specified here must be defined in a MASTER-LU61-LPAP statement. If you specify BUNDLE, this LPAP becomes a slave LPAP of the specified master LPAP.

```
MASTER-LU61-LPAP master, ...  
LPAP slave-lpap, BUNDLE=master, ...
```

CONs of LPAPs belonging to an LU6.1-LPAP bundle

- No physical connections (CONs) can be assigned to a master LPAP. This means that it cannot be specified as the LPAP in a CON statement. The master LPAP always uses the connections assigned to the slave LPAPs.

Distribution of messages

For details, refer to the section [“Distributing messages” on page 107](#).

Display in the KB header

For details, refer to the section [“Information displayed in the KB header” on page 108](#).

4.1.5 LU6.1-LPAP bundles of a standalone application with a UTM cluster application

Note the following when generating LU6.1 communication between a standalone partner application and a UTM cluster application:

- A partner application must generate one LPAP with a specific number of sessions and connections for each node of the UTM cluster application with which it wants to communicate.
- To address the UTM cluster application, an LU6.1-LPAP bundle whose slave LPAPs are assigned to the cluster node should be generated in the partner application (see the [section “MASTER-LU61-LPAP – Define the master LPAP of an LU6.1-LPAP bundle” on page 366](#)).
- In the UTM cluster application, more sessions (LSES) than connections (CON) must be generated for the LPAP that represents the partner application: One session per cluster node must be generated for each connection.
- Each cluster node requires only exactly the number of connections assigned to each LPAP in the partner application for the corresponding LPAP. However, because all cluster nodes have identical generations, the sessions for all the LPAPs of the partner application must be generated in every cluster node.

Connection from an LU6.1 partner application generated in this way must always be established by the standalone partner application, and never by the UTM cluster application. In this case, do not generate the LU6.1 connections in the UTM cluster application using `autoconnect`, i.e. specify `CONNECT=NO` (default) in the `SESCHA` statement (see also [page 470](#)).

For administration purposes, an LU6.1 session can also be established by the UTM cluster application, although this is only possible using the `KDCLSES` command or by modifying the `LSES` object for programmed administration. To do this, the administrator must select a session (LSES) whose remote session (RSES) in the partner application is assigned to the LPAP that represents the local node application in the remote application.

Example:

The example below shows a generation in which the standalone application SA on the host HOSTSA is linked to the UTM cluster application CA on the cluster nodes NODECAX, NODECAY and NODECAZ. 4 connections between the standalone application and each of the node applications are generated. A MASTER-LU61-LPAP is generated for the LPAPs that represent the node applications in the standalone application. This represents the UTM cluster application.

Standalone application SA on HOSTSA

```

BCAMAPPL SA11
BCAMAPPL SA12
MASTER-LU61-LPAP MLPAPCA
LPAP LPAPCAX, SESCHA=SESCHCA-
    , BUNDLE=MLPAPCA
LPAP LPAPCAY, SESCHA=SESCHCA-
    , BUNDLE=MLPAPCA
LPAP LPAPCAZ, SESCHA=SESCHCA-
    , BUNDLE=MLPAPCA
SESCHA SESCHCA, CONTWIN=NO, PLU=YES
CON CA11, PRONAM=NODECAX -
    , BCAMAPPL=SA11, LPAP=LPAPCAX
CON CA12, PRONAM=NODECAX -
    , BCAMAPPL=SA11, LPAP=LPAPCAX
CON CA11, PRONAM=NODECAX -
    , BCAMAPPL=SA12, LPAP=LPAPCAX
CON CA12, PRONAM=NODECAX -
    , BCAMAPPL=SA12, LPAP=LPAPCAX
CON CA11, PRONAM=NODECAY -
    , BCAMAPPL=SA11, LPAP=LPAPCAY
CON CA12, PRONAM=NODECAY -
    , BCAMAPPL=SA11, LPAP=LPAPCAY
CON CA11, PRONAM=NODECAY -
    , BCAMAPPL=SA12, LPAP=LPAPCAY
CON CA12, PRONAM=NODECAY -
    , BCAMAPPL=SA12, LPAP=LPAPCAY
CON CA11, PRONAM=NODECAZ -
    , BCAMAPPL=SA11, LPAP=LPAPCAZ
CON CA12, PRONAM=NODECAZ -
    , BCAMAPPL=SA11, LPAP=LPAPCAZ
CON CA11, PRONAM=NODECAZ -
    , BCAMAPPL=SA12, LPAP=LPAPCAZ
CON CA12, PRONAM=NODECAZ -
    , BCAMAPPL=SA12, LPAP=LPAPCAZ

```

**UTM cluster application CA on
NODECAX/Y/Z**

```

BCAMAPPL CA11
BCAMAPPL CA12
LPAP LPAPSA, SESCHA=SESCHSA
SESCHA SESCHSA, CONTWIN=YES, PLU=NO
CON SA11, PRONAM=HOSTSA -
    , BCAMAPPL=CA11, LPAP=LPAPSA
CON SA11, PRONAM=HOSTSA -
    , BCAMAPPL=CA12, LPAP=LPAPSA
CON SA12, PRONAM=HOSTSA -
    , BCAMAPPL=CA11, LPAP=LPAPSA
CON SA12, PRONAM=HOSTSA -
    , BCAMAPPL=CA12, LPAP=LPAPSA

```

```

LSES SAA2CAX, RSES= CA12SA1-
, LPAP=LPAPCAX
LSES SAB2CAX, RSES= CA12SA2-
, LPAP=LPAPCAX
LSES SAC2CAX, RSES= CA12SA3-
, LPAP=LPAPCAX
LSES SAD2CAX, RSES= CA12SA4-
, LPAP=LPAPCAX

LSES SAA2CAY, RSES= CA22SA1-
, LPAP=LPAPCAY A
LSES SAB2CAY, RSES= CA22SA2-
, LPAP=LPAPCAY
LSES SAC2CAY, RSES= CA22SA3-
, LPAP=LPAPCAY
LSES SAD2CAY, RSES= CA22SA4-
, LPAP=LPAPCAY

LSES SAA2CAZ, RSES= CA32SA1-
, LPAP=LPAPCAZ
LSES SAB2CAZ, RSES= CA32SA2-
, LPAP=LPAPCAZ
LSES SAC2CAZ, RSES= CA32SA3-
, LPAP=LPAPCAZ
LSES SAD2CAZ, RSES= CA32SA4-
, LPAP=LPAPCAZ

LSES CA12SA1, RSES= SAA2CAX-
, LPAP=LPAPSA
LSES CA12SA2, RSES= SAB2CAX-
, LPAP=LPAPSA
LSES CA12SA3, RSES= SAC2CAX-
, LPAP=LPAPSA
LSES CA12SA4, RSES= SAD2CAX-
, LPAP=LPAPSA

LSES CA22SA1, RSES= SAA2CAY-
, LPAP=LPAPS
LSES CA22SA2, RSES= SAB2CAY-
, LPAP=LPAPSA
LSES CA22SA3, RSES= SAC2CAY-
, LPAP=LPAPSA
LSES CA22SA4, RSES= SAD2CAY-
, LPAP=LPAPSA

LSES CA32SA1, RSES= SAA2CAZ-
, LPAP=LPAPSA
LSES CA32SA2, RSES= SAB2CAZ-
, LPAP=LPAPSA
LSES CA32SA3, RSES= SAC2CAZ-
, LPAP=LPAPSA
LSES CA32SA4, RSES= SAD2CAZ-
, LPAP=LPAPSA

```

If you want to establish an LU6.1 session to the standalone application SA in the node application CA on the node NODECAY, this can only be done via the sessions CA22SA1, CA22SA2, CA22SA3 and CA22SA4 because their remote sessions SAA2CAY, SAB2CAY, SAC2CAY and SADA2CAY are assigned to the LPAP LPAPCAY.

Example:

```
KDCLSES LSES=CA22SA1, ACT=C
```

4.2 Distributed processing via the OSI TP protocol

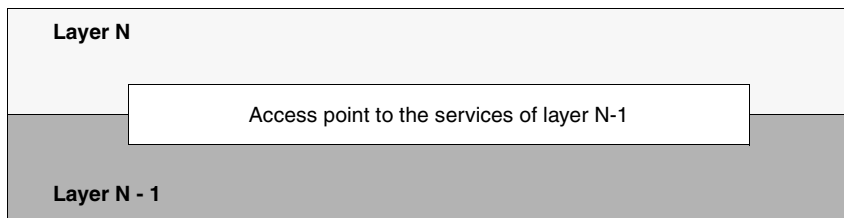
Before discussing the rules and recommendations for generating UTM applications with distributed processing via OSI TP, a number of OSI terms relevant for configuration are explained below. The OSI terms in this section are shown in *italics*.

4.2.1 OSI terms

If two partners wish to communicate with each other, the rules they must observe and the services they must provide have been standardized in the OSI protocol (Open System Inter-connection).

ISO (International Organization for Standardization) has also developed the *OSI reference model*, in which the various communication tasks are distributed over *seven layers (instances)*. The services to be provided by each layer are clearly defined. The seven layers form a hierarchical structure, where each layer can access the *services* of the underlying layer. These services are made available to the overlying layer at *service access points*.

During communication, the application accesses the services of the communication system via one of these access points:



If two applications wish to communicate and exchange data, they must be linked via a *transport connection*. A transport connection can only be established between two *addressable units* in the network. It must therefore be possible to identify each application by means of an *address* which is unique throughout the network.

In the OSI world, addresses are assigned to service access points rather than applications. Within the network, the application is thus identified by means of the address of the access point via which it communicates.

Each service access point is assigned an address which is unique throughout the network. This consists of a *selector* and the address of the underlying access point. The following diagram shows the format of addresses in the OSI reference model.

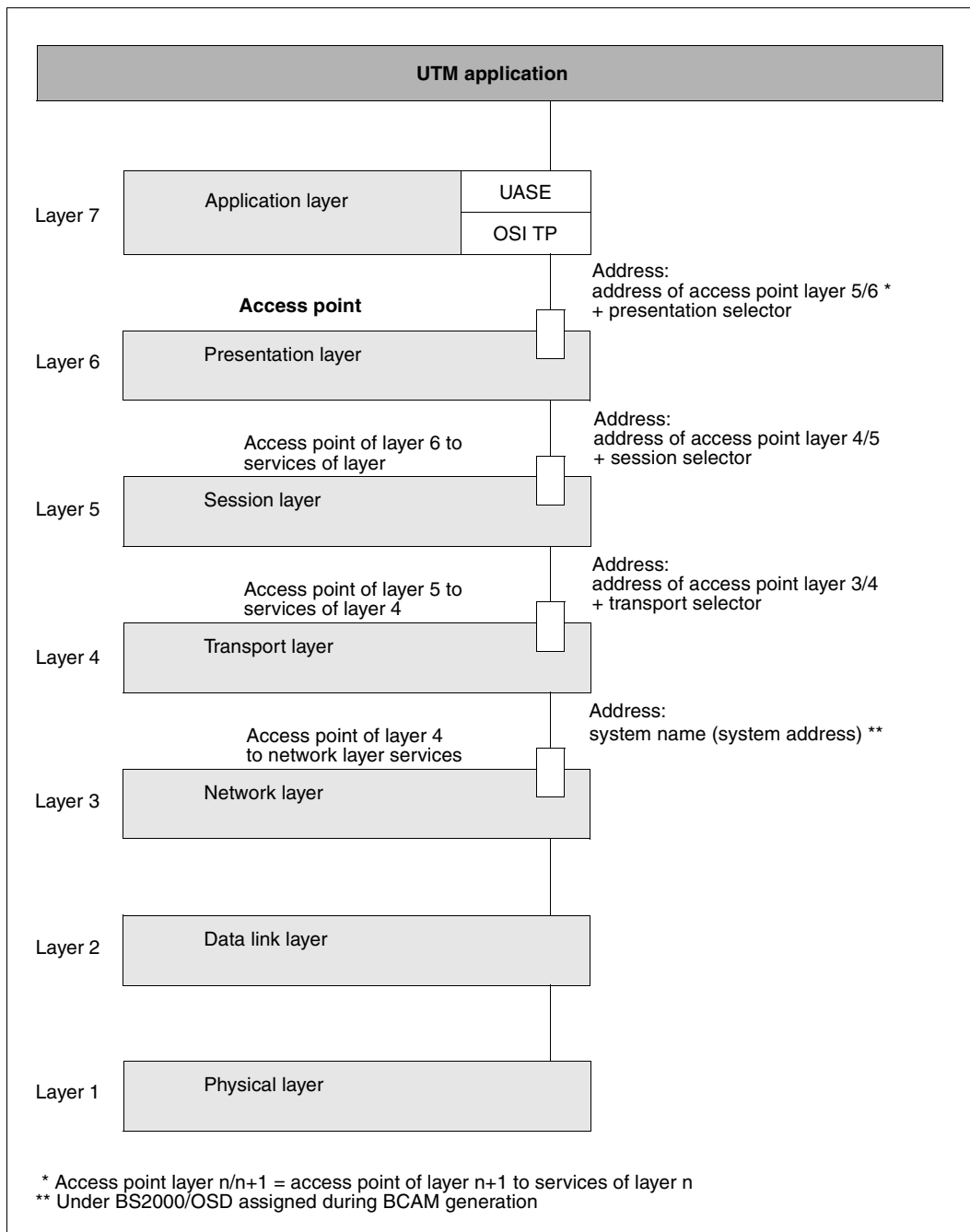


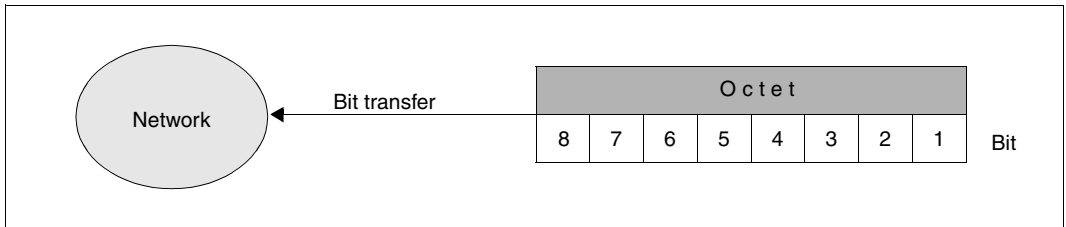
Figure 8: Addresses of the service access points in the OSI reference model

In order to communicate with other applications in the network, a UTM application links to a service access point. This link is generated using the ACCESS-POINT statement. The UTM application can then be accessed by its partners in the network via the address of this access point.

The format of the access point address via which the application is accessed depends on the access point hierarchy. If the UTM application communicates on the basis of OSI TP, it links to an access point to the services of layer 6. The address of the access point in the local system consists of the *transport selector* (selector of the *transport layer*), the *session selector* (selector of the *session layer*), and the *presentation selector* (selector of the *presentation layer*). The network address of the access point thus comprises the network address of the system and the address of the access point in the local system.

The selectors of the individual layers must be unique in the local system. The system address is unique throughout the entire network. When defining the access point address, you must consult your network administrator.

The selectors consist of *octets*. An octet is a byte (8 bits) in which the bit numbering and thus the order in which bits are transferred is fixed.



Order of bit transfer in an octet

It is possible to establish several *parallel connections* (also known as *associations*) to a certain communication partner. All connections to a remote partner are generated in a **single** OSI-CON statement. The OSI-LPAP statement can be used to define the number of parallel connections to a particular partner. Each individual connection must be assigned an *association name*, which is unique throughout the local system. The names of associations with a remote partner are also generated in the OSI-LPAP statement.

Each connection between two partners is managed by one of the partners. This partner is known as the *contention winner*, while the other partner is referred to as the *contention loser*. Jobs can be initiated by both partners. If both partners submit a job at the same time, priority is given to the contention winner. The contention winner of a connection should be the communication partner that starts jobs most frequently.

If several parallel connections exist between two partner, it is not necessary to define which partner is the contention loser and which partner is the contention winner for each connection. You simply specify the number of connections for which the individual partners act as the contention winner (OSI-LPAP statement). The partner that starts jobs most frequently should be defined as the contention winner.

openUTM supports *TPSU-title* (**T**ransaction **P**rocessing **S**ervice **U**ser). This is an OSI TP user which provides certain services within an application. In openUTM this is the sequence of program units which form a service. A TPSU-title is a unique name within the application. In openUTM it is the TAC name of the first program unit of a service. The *initiating TPSU-title* is the TPSU-title of the job submitter and the *recipient TPSU-title* is the TPSU-title of the job receiver.

openUTM supports the *application entity title* (AET) defined by ISO. This is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET in order to establish a connection. In openUTM, the AET is specified for information purposes, but it is not used for addressing the partner.

The *application context* to be used for communication purposes must be defined for each remote partner with which the UTM application wishes to communicate via the OSI TP protocol.

The OSI terms *application entity title* and *application context* are described in further detail in the following sections.

Application entity title (AET)

In the OSI world, communication partners are represented by application entities. An application entity is an addressable unit in layer 7 of the OSI reference model (application layer). An example would be the access point of a UTM application, via which an OSI TP communication partner can link to the UTM application. In the OSI TP standard, each application entity is assigned an application entity title, which can be used to uniquely address the application entity in the OSI network.

The ISO standard defines two forms of AET: the directory form and the object identifier form. The latter is supported by openUTM. This is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET in order to establish a connection. In the case of homogeneous communication between UTM and UTM, the AET is also specified, but is not used for addressing the partner. It consists of two parts:

- the application process title (APT)
- the application entity qualifier (AEQ)

Application process title (APT)

The APT is used to identify the application. In accordance with the ISO standard, it must be unique globally (i.e. worldwide). For this reason, it should be assigned and registered by a standardization body, e.g. in Germany this is the Deutsche Gesellschaft for Warenkennzeichnung GmbH (DGWK = Germany company for registering trademarks).

An APT in object identifier form consists of up to 10 components:

(component1,component2,...,component10)

Some of the values for *component1* through *component10* have been standardized. Here, symbolic names have been assigned to certain numbers. The value range for *component2* depends on the value for *component1*. The table below lists the symbolic names and value ranges supported by openUTM:

component1	0:CCITT	1:ISO	2:JOINT-ISO-CCITT
component2	0:RECOMMENDATION 1:QUESTION 2:ADMINISTRATION 3:NETWORK-OPERATOR	0:STANDARD 1:REGISTRATION-AUTHORITY 2:MEMBER-BODY 3:IDENTIFIED-ORGANIZATION	
	Value range: 0 - 39	Value range: 0 - 39	Value range: 0 - 67 108 863
component3 through component10	Value range: 0 - 67 108 863	Value range: 0 - 67 108 863	Value range: 0 - 67 108 863

The APT specified in openUTM need not be assigned by a standardization body, i.e. it is freely selectable. However, it must meet the following two requirements:

- it must be unique within the network
- it must contain permitted values, as shown in the table above

Application entity qualifier (AEQ)

The AEQ identifies an access point within an application. You can only assign AEQs to the access points of an application if you have assigned an APT to the application itself.

The AEQ is a positive integer between 0 and 67108863.

The AEQ must be unique within the application, i.e. the application must not contain two access points with the same AEQ. However, it is not necessary to assign an AEQ to all access points in the application.

When there are parallel associations and a connection is being established, the AEQ is checked to see if it is the same one as used for the first association established.

Application context

The application context to be used for communication purposes must be coordinated with each partner application with which your local application wishes to communicate via the OSI TP protocol.

The application context must be explicitly defined for each partner application. It determines the rules governing the transfer of messages between the local application and partner application. openUTM supports the following predefined application contexts:

- UDTAC
- UDTDISAC
- XATMIAC
- UDTCCR
- UDTSEC
- XATMICCR

If you are not using one of the application contexts listed above you can use the APPLICATION-CONTEXT statement which is described on [page 285](#) to generate further application contexts.

All the involved partners must agree the following when using an application context:

- An *abstract syntax*, which defines how the user data is encoded for transfer. By default, openUTM supports the following abstract syntaxes:
 - UDT (Unstructured Data Transfer)
 - XATMI
 - CCR
 - UTMSEC

See also the ABSTRACT-SYNTAX statement on [page 275](#).

- A *transfer syntax*, which defines the format in which the user data is transferred. By default, openUTM supports the transfer syntax Basic Encoding Rules (BER).

See also the TRANSFER-SYNTAX statement on [page 528](#).

Both communication partners must generate the same abstract syntaxes as the application context used for communication. If the application context generated locally is not identical to that generated in the partner, openUTM rejects any attempts to establish the association with a corresponding message.

You only need to use the ABSTRACT-SYNTAX, TRANSFER-SYNTAX and APPLICATION-CONTEXT statements if you are not using any of the standard application contexts made available by openUTM.

4.2.2 Generation procedure for distributed processing based on OSI TP

The following KDCDEF statements are provided for generating the communication partners of an application and the connections to these partners:

Statement	Function
ABSTRACT-SYNTAX	Define the abstract syntax for the user data: <ul style="list-style-type: none"> – assign a unique object identifier – assign the transfer syntax for data transfer
ACCESS-POINT	Define the name and address of the local OSI TP access point: <ul style="list-style-type: none"> – define the application entity qualifier (AEQ) of the local application (address component of the application entity title)
APPLICATION-CONTEXT	Define the application context for communication with the partner application: <ul style="list-style-type: none"> – assign the abstract syntax for the user data – assign a unique object identifier
LTAC	Assign local TAC names for services in the partner application, under which these services are then started locally
MASTER-OSI-LPAP	Define the name and properties of the master LPAP in a OSI-LPAP bundle (see page 105)
OSI-CON	Define connections between the local application and the remote partner and assign these to the OSI-LPAP partner: <ul style="list-style-type: none"> – specify a local OSI TP access point – specify the network address of the partner application
OSI-LPAP	Define an OSI-LPAP partner as the logical access point for the partner application: <ul style="list-style-type: none"> – specify the application entity title (i.e. APT and AEQ) of the partner application – specify the application context of the partner application – define the number of (parallel) connections to the partner and the names of these connections – define the number of connections to be established automatically when the application is started – define the number of connections for which the local application is to act as the contention winner – define the access rights of the partner application in the local application – define the administration authorization level of the partner application – define maximum values for the message queue of the OSI-LPAP partner – define the status of the OSI-LPAP partner on connection setup – if necessary, make the OSI-LPAP a slave LPAP of a OSI-LPAP bundle and specify the associated master LPAP

Statement	Function
TRANSFER-SYNTAX	Define the transfer syntax for data transfer: – assign a unique object identifier
UTMD	Define global values and the address of the local UTM application: – define the application process title (APT) (address component of the application entity title) – define the maximum waiting time for establishing an association – define the maximum waiting time for confirmation of asynchronous messages
X/W X/W	The following additional parameters are available under Unix systems and Windows systems:
X/W MAX XAPTPSHMKEY	Key for the XAPTP shared memory segment.
X/W MAX OSISHMKEY	Define an authorization key for the OSS shared memory segment
X/W MAX OSI-SCRATCH-AREA	Define the size of the working area for dynamic data storage

To allow for communication based on the OSI TP protocol, you must perform the following steps:

- Generate the application entity title (AET)

The statement `UTMD...,APPLICATION-PROCESS-TITLE=` is used to define the application process title (APT) as the address component of the AET for your application. A remote partner that requires AETs must know this APT in order to establish a connection.

The application entity title is assigned to the OSI-LPAP partner. In the OSI-LPAP statement, use the `APPLICATION-PROCESS-TITLE=` operand to specify the APT and the `APPLICATION-ENTITY-QUALIFIER=` operand to specify the AEQ of the access point for the partner application. The AEQ must already be generated for the access point in the remote partner application.

Using the statement `ACCESS-POINT...,APPLICATION-ENTITY-QUALIFIER=`, define the application entity qualifier (AEQ) as the address component of the AET for the access point of the local application. A partner application must know the AEQ of the access point via which communication takes place with the local application.

- Define the application context for communication with the partner application

If you do not wish to work with one of the default application contexts listed on [page 98](#), you can generate the application context to be used for communication with the partner application using the `APPLICATION-CONTEXT` statement. This involves assigning defined abstract syntaxes and a unique object identifier to the application context.

The `ABSTRACT-SYNTAX` statement serves to specify the abstract syntax used for the transfer of user data and to assign a unique object identifier to this abstract syntax.

The transfer syntax (which defines how the user data is to be encoded and decoded for data transfer) is also defined using the ABSTRACT-SYNTAX statement. The transfer syntax is identified by means of a unique object identifier.

- Define an access point to the OSI TP services for your UTM application, so that your application can be addressed during communication based on OSI TP.

The ACCESS-POINT statement is used to specify the address of the access point within the local system, and to assign a symbolic name for addressing the access point in the local UTM application.

The address defined in ACCESS-POINT must be unique within the UTM application and within the local system (under BS2000/OSD for each host). When defining the access point address, you must therefore consult your system or network administrator.

A partner application that wishes to communicate with the local application on the basis of the OSI TP protocol identifies the local UTM application using the access point address and the network address of your system. The network address of the access point must be specified when generating the remote partner applications.

- Define a logical access point (OSI-LPAP partner) for each partner application and the connections between the local application and partner application

This involves the definition of an OSI-LPAP statement and an OSI-CON statement. An OSI-LPAP statement must be generated for each partner application. The OSI-CON statement defines the connections between the UTM application and the communication partner. The definition is generated via the two access points (in the local application and in the partner application) between which connections are to be established. The OSI-CON statement is used to specify the network address of the remote access point and the name of the local access point (defined in ACCESS-POINT). The OSI-LPAP statement is used to define the number and names of parallel connections (associations) to the partner application. The address of the remote access point must match that of the access point generated in the partner application.

If connections are to be established automatically as soon as both communication partners are available (i.e. started), this must be specified when generating **both** partners. In openUTM, this is achieved using the statement OSI-LPAP...,CONNECT=. The partner started last then establishes the connection. With OSI-LPAP...,CONTWIN=, you can also define the number of connections to the communication partner for which the local application is to act as the contention winner.

- Assign local transaction codes to the services of the partner applications, which are then used to address remote services in the local application

Each of these transaction codes is defined using an LTAC control statement. The transaction code can be assigned uniquely to the partner application via the LPAP partner using LTAC..., LPAP=*osi-lpapname*. In addition, you must specify the transaction code of the program unit in the partner application using LTAC..., RTAC=. Ask the operator of the partner application for this transaction code. If the name and type of the remote TAC are specified incorrectly, this is not detected by the KDCDEF generation tool, since KDCDEF does not have any information on the configuration of the partner application. The error is not detected until the local application requests this LTAC.

You can also perform the following steps for communication based on OSI TP:

- Define global values for all connections from the application to communication partners

Using the UTMD statement, you can restrict and define the time spent waiting for confirmation from the communication partner, and specify the total number of jobs submitted to partner applications that can be processed simultaneously in the local application (via OSI TP and LU6.1). By defining appropriate limit values, you can prevent connections from becoming blocked or from being terminated prematurely. You can also ensure that all tasks of the application are occupied by jobs from remote applications. The values defined here also apply for communication based on LU6.1.

- Generate replacement connections

Replacement connections can be generated by issuing two OSI-CON statements for the same connection. They are used to interact alternatively with various partners without having to take this into consideration in the program units. The two partners may be located in different systems. The replacement connection is generated by assigning two OSI-CON statements with different partner addresses (remote access point addresses) to an OSI-LPAP statement, thereby allocating two different partners to the same logical access point (LPAP partner). However, both connections to the alternative partner applications must not be activated simultaneously. For this reason, ACTIVE=YES may only be specified in one OSI-CON statement. You can switch to the replacement connection to the alternative partner application using the KDCLPAP administration command.

- Define data access control for services of the partner application

Using the statement LTAC..., LOCK= or LTAC..., ACCESS LIST=, you can secure a partner application service with a lock code. This service is then available locally only to those program units that are running under a user ID (KCBENID) and have been started by a client (KCLOGTER) that have the appropriate authorization.

- Define data access control for services of the local application

If you wish to restrict a partner application's access to certain services of the local application, you can secure critical services with a lock code or an access list. With the KSET statement, you can define a key set containing the key codes for services that can be accessed by the partner application. This key set is assigned to the logical access point of the partner application using the statement `OSI-LPAP...,KSET=`. The partner application can then call only those TACs which are either not secured or for which the partner application has the appropriate authorization.

- Assign administration authorization for TACs of the partner application

In the OSI-LPAP statement, you can define whether the partner is to be granted administration authorization in the local application. The authorization level is defined using `OSI-LPAP...,PERMIT=`.

B

- Assign UTM SAT administration authorization for TACs of the partner application

B

Using `OSI-LPAP...,PERMIT=` you also can define whether the partner is to be granted UTM SAT administration authorization in the local application.

B

The diagram on [page 104](#) summarizes the areas in which the generation of the local application must be coordinated with the generation of the partner application. A standard application context generated by openUTM is used. You must not therefore enter an `APPLICATION-CONTEXT` statement.

For more information on generating applications with distributed processing, in particular via OSI TP, refer to the example generation „ComfoTravel“ on [page 550ff.](#)

Local application Reservation Management Service	Partner application Travel Agency
. .	. .
ACCESS-POINT ACRMS, P-SEL=(C'PRMS',ASCII), (TS) S-SEL=(C'SRMS',ASCII), (TS) T-SEL=C'RMS', (TS) APPLICATION-ENTITY-QUALIFIER=11 (1)	OSI-CON CNRMS, P-SEL=(C'PRMS',ASCII), (TS) S-SEL=(C'SRMS',ASCII), (TS) T-SEL=C'RMS', (TS) N-SEL=C'HOST001', (TS) LOCAL-ACCESS-POINT=ACTRAVEL, OSI-LPAP=LPRMS
OSI-CON CNAGENCY, P-SEL=*NONE, S-SEL=*NONE, T-SEL=C'TRAVEL', (TS) N-SEL=C'HOST002', (TS) LOCAL-ACCESS-POINT=ACRMS, OSI-LPAP=LPAGENCY	ACCESS-POINT ACTRAVEL, P-SEL=*NONE, S-SEL=*NONE, T-SEL=C'TRAVEL', (TS) APPLICATION-ENTITY-QUALIFIER=21 (2)
OSI-LPAP LPAGENCY, APPLICATION-CONTEXT=UDTCCR, APPLICATION-ENTITY-QUALIFIER=21, (2) APPLICATION-PROCESS-TITLE=(1,2,3,20), (3) ASSOCIATIONS=4, ASSOCIATION-NAMES=AGENCY, CONNECT=2, CONTWIN=1	OSI-LPAP LPRMS, APPLICATION-CONTEXT=UDTCCR, APPLICATION-ENTITY-QUALIFIER=11, (1) APPLICATION-PROCESS-TITLE=(1,2,3,10), (5) ASSOCIATIONS=4, ASSOCIATION-NAMES=RMS, CONNECT=2, CONTWIN=3
LTAC LTAGENCY, LPAP = LPAGENCY, RTAC = TCAGENCY, (4) WAITTIME=(10,30)	TAC TCAGENCY, (4) PROGRAM=PRAGENCY
UTMD MAXJR=200, APPLICATION-PROCESS-TITLE=(1,2,3,10), (5) CONCTIME=25, PTCTIME=0	UTMD MAXJR=200, APPLICATION-PROCESS-TITLE=(1,2,3,20), (3) CONCTIME=25, PTCTIME=0
.

(n) specifies that values must correspond across both OSI TP generations.

(TS) means the values must also correspond to those of the transport system or network generation.

Figure 9: Coordination during the generation of OSI TP applications

4.2.3 OSI-LPAP bundles

OSI-LPAP bundles allow automatic distribution of messages over multiple OSI-LPAP partners. If a UTM application exchanges a large number of messages with a partner application, it may make sense in terms of the load balancing to start several instances of the partner application and distribute the messages among the separate instances. In a OSI-LPAP bundle, openUTM takes care of distributing the messages to the instances of the partner application. To achieve this, the program units in the APRO call must address the MASTER-OSI-LPAP.

One case of an application with this type of message distribution when a UTM application communicates via BeanConnect with a J2EE application server. If the application server is run as a cluster application, then the messages sent to the application server should be distributed among the separate instances of the cluster (see also the "BeanConnect for openUTM" manual).

A further application scenario is communication from a standalone UTM application to a UTM cluster application. This allows messages to the UTM cluster application to be distributed across the individual node applications.

i Connecting a standalone UTM application to a UTM cluster application via an OSI-LPAP bundle works in the same way as communication between two standalone UTM applications with OSI-LPAP bundles. No special issues related to clusters need to be observed.

An OSI-LPAP bundle consists of one master LPAP and several slave LPAPs. The slave LPAPs are assigned to the master LPAP during generation. In this case, OSI-CONs that belong to different slave LPAPs address the various partner applications.

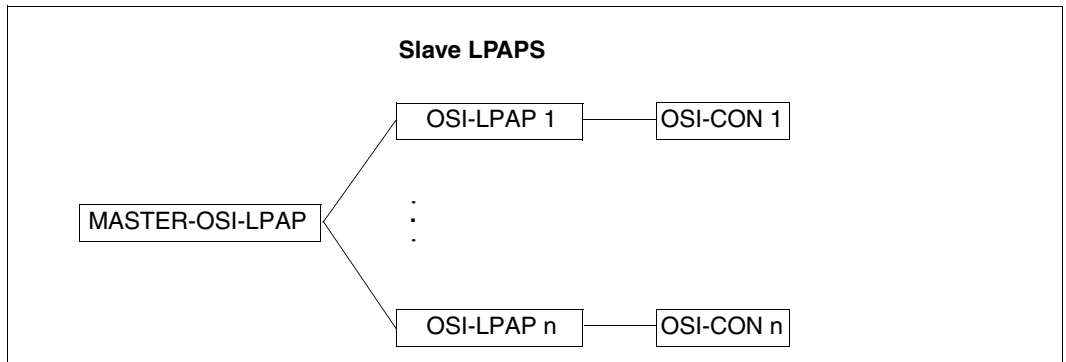


Figure 10: Example of a OSI-LPAP bundle

Generating OSI-LPAP bundles



MASTER-OSI-LPAP statement on [page 367](#)

Defines the name and properties of the master LPAP in a OSI-LPAP bundle:

- *master-lpap-name*
Name for the master LPAP.
- APPLICATION-CONTEXT=
Application context to be used for the communication with the remote partner.
- STATUS=
Specifies whether messages can be sent to this LPAP bundle.



OSI-LPAP statement on [page 426](#)

The following properties must be specified for the generation of a slave LPAP:

- *lpap-name*
Name of the slave LPAP.
- BUNDLE=master-lpap-name
Name of the master LPAP. The master LPAP specified here must be defined in a MASTER-OSI-LPAP statement. If you specify BUNDLE, this OSI-LPAP becomes a slave LPAP of the specified master LPAP.

MASTER-OSI-LPAP *master*, ...
OSI-LPAP *slave-lpap*, BUNDLE=*master*, ...
- APPLICATION-CONTEXT=
Application context to be used for the communication with the remote partner.

All slave LPAPs of a LPAP bundle must be assigned to the same application context as the master LPAP.

OSI-CONs of LPAPs in an OSI-LPAP bundle

- Physical connections (OSI-CONs) must not be assigned to master LPAP. This means it may not be specified as an OSI-LPAP in a OSI-CON statement. The master LPAP always uses the connections assigned to the slave LPAPs.
- All OSI-CONs of all slave LPAPs of a LPAP bundle must be assigned to the same local ACCESS-POINT.

Distributing messages



The following information on distributing messages applies equally to LU6.1 and OSI TP.

Program units can address a slave LPAP as well as a master LPAP with the APRO call. APRO calls to a slave LPAP are not distributed by openUTM. APRO calls to a master LPAP are distributed as follows by openUTM:

- openUTM addresses the slave LPAPs in sequence using APRO calls sent to a master LPAP.
 - openUTM always attempts in this case to find a slave LPAP to which a connection has already been established and, if a queued message is to be sent to the partner (APRO AM), whose queue level has not been reached yet.
 - If the first APRO call to a master LPAP in a transaction is APRO DM, then openUTM only returns the return code 40Z/KD10 when there is no connection to any slave LPAP.
 - If the first APRO call to a master LPAP in a transaction is APRO AM, then openUTM only selects a slave LPAP with a cleared connection when there is no connection yet to any of the slave LPAPs. In this case the connection is initiated for the slave LPAP.
 - When searching for a slave LPAP with an established connection, a connection is initiated for every slave LPAP found that does not have a connection yet.
- All APRO calls sent to the MASTER-LPAP in a single transaction address the same slave LPAP.

For this reason, an APRO call for a second message to a partner application may be rejected if, for example, the queue level for the slave LPAP has been exceeded or the connection has been lost in the meantime.
- Messages that have already been assigned to a slave LPAP are not reassigned in sequence to another slave LPAP any more. If the connection for dialog messages is lost after the APRO call, then the dialog message is rejected just like for "normal" LPAPs and the transaction is reset, if necessary.

Information displayed in the KB header



The following information on the display in the KB header applies equally to LU6.1 and OSI TP.

In services started for received messages, openUTM always displays the name of the LTERM or (OSI-)LPAP through which the message was received in the KB header.

The following therefore applies for LPAP bundles:

In services started for messages received through a slave LPAP, the name of this slave LPAP is displayed in the KB header and **not** the name of the master LPAP.

With INIT PU you can obtain information on whether the (OSI-)LPAP in the KB header is the slave LPAP of an LPAP bundle as well as the name of the master LPAP.

4.3 Coordinating the UTM and BCAM generations (BS2000/OSD)

B During distributed processing via LU6.1 and OSI TP network connections are required to
 B enable applications to communicate with each other. openUTM uses the services of the
 B BCAM transport system for these network connections.

B To allow you to establish these network connections, addresses must be assigned to both
 B communication partners. These addresses must be unique throughout the network. The
 B network type required for communication is defined by means of appropriate entries in
 B KDCDEF generation (BCAMAPPL T-PROT= statement) and in BCAM generation. It is
 B important to note that no distinction is made in openUTM between ISO and TCP/IP
 B networks (ISO and RFC1006 entries for the T-PROT parameter are synonymous). This
 B distinction must be made in BCAM generation when defining connections between partici-
 B pating computers.

B It is not usually necessary to make any entries in BCAM for a UTM application.

B In the case of a connection via RFC1006, BCAM uses the number "102" as the partner's
 B listener port number by default. However, if a different port number is used, e.g. because
 B the transport system of the partner cannot use port number 102, then a BCPMAP entry must
 B be created for the remote application (SUBFUNCT=GLOBAL). This BCPMAP entry is
 B created as shown below (see also the "BCAM User Guide"):

```
B /BCMAP FUNCT=DEFINE , SUBFUNCT=GLOBAL , NAME=name-of-partner-application , -
B /      ES=partner-processor-name , PPORT#=listener-port-no , -
B /      PTSEL-I=tselector-of-partner-application
```

4.4 Providing address information for the CMX transport system (Unix systems and Windows systems)

X/W In the case of distributed processing via LU6.1 or OSI TP and when connecting clients of type PTYPE=APPLI and UPIC-R, openUTM uses the CMX transport access system (Unix systems and Windows systems). When communicating using CMX, the connection is established using TCP/IP-RFC1006. openUTM obtains the IP addresses of the partner computer when started from the corresponding database, e.g. from the hosts file or the domain name server.

X/W Port number 102 for TCP/IP connections

X/W The applications are accessed via port numbers in connections via TCP/IP-RFC1006. You must note the following, regardless of how you provided the address information:

- X/W – Only port number 102 can be used for local transport system endpoints (BCAMAPPL, ACCESS-POINT) in UTM applications on Unix systems and Windows systems as well as for partner applications and clients running on Unix systems and Windows systems.
- X/W – In UTM applications running on Unix systems and Windows systems, you can only use port number 102 for partner applications and clients on BS2000/OSD.

4.4.1 Providing address information with KDCDEF

X/W The address information is stored in the PRONAM, N-SEL, LISTENER-PORT operands, as well as in the T-PROT and TSEL-FORMAT operands. You must note the following:

- X/W – You **must** always specify the first TCP/IP host name in the database (hosts file, DNS, ...) for PRONAM or N-SEL or an assigned mapped host name (see [section “Using mapped host names \(Unix systems and Window systems\)” on page 122](#)). For actively established connections, openUTM determines the name from the IP address, and for passively established connections the IP address from the name. You must not specify any other names, i.e. aliases, listed in the database (hosts file, DNS,...) here.
- X/W – You should always enter a port number for LISTENER-PORT. The port number must always match the port number used by the communication partner.
- X/W – You must always specify RFC1006 for T-PROT.
- X/W – It is recommended to always enter data for the TSEL-FORMAT operand. If you do not specify anything there, then KDCDEF assigns one of the following values, depending on the OPTION statement:
 - X/W – for CHECK-RFC1006=YES, a default value based on the set of characters in the name of the corresponding application or partner
 - X/W – for OPTION CECK-RFC1006=NO, the invalid value 'U' or '?' (= undefined)

X/W OSI TP connection

X/W In the following example, port number 10000 is used in the local application and port number
 X/W 12000 is used in the remote application. The remote application is running on the computer
 X/W named CENTRAL1.

```
X/W ACCESS-POINT BSPOSITP -
X/W ,LISTENER-PORT=10000 -
X/W ,T-PROT=RFC1006 -
X/W ,TSEL-FORMAT=T -
X/W ,P-SEL=... ,S-SEL=... ,T-SEL=... -
X/W ....
```

```
X/W OSI-CON OSICON01 -
X/W ,LOCAL-ACCESS-POINT=BSPOSITP -
X/W ,LISTENER-PORT=12000 -
X/W ,T-PROT=RFC1006 -
X/W ,N-SEL=CENTRAL1 -
X/W ,P-SEL=... ,S-SEL=... ,T-SEL=... -
X/W ....
```

X/W The specifications for P-SEL, S-SEL and T-SEL need to match the values specified in the
 X/W generation of the partner application (see the sample on [page 104](#)).

X/W LU6.1 connection

X/W In the following example, port number 10010 is used in the local application and port number
 X/W 12010 is used in the remote application. The remote application is running on the computer
 X/W named CENTRAL2.

```
X/W BCAMAPPL BSPLU61 -
X/W ,LISTENER-PORT=10010 -
X/W ,T-PROT=RFC1006 -
X/W ,T-SEL-FORMAT=T -
X/W ....
```

```
X/W CON LU61PART -
X/W ,BCAMAPPL=BSPLU61 -
X/W ,PRONAM=CENTRAL2 -
X/W ,LISTENER-PORT=12010 -
X/W ,T-PROT=RFC1006 -
X/W ,TSEL-FORMAT=T -
X/W ....
```

X/W **PTYPE=APPLI connection**

X/W In the following example, port number 10020 is used in the local application and port number
X/W 12020 is used in the remote application. The remote application is running on the computer
X/W named CENTRAL3.

```
X/W BCAMAPPL BSPAPPLI -
X/W ,LISTENER-PORT=10020 -
X/W ,T-PROT=RFC1006 -
X/W ,T-SEL-FORMAT=T -
X/W . . . .
X/W
X/W PTERM APPLPART -
X/W ,PTYPE=APPLI,
X/W ,BCAMAPPL=BSPAPPL -
X/W ,PRONAM=CENTRAL3 -
X/W ,LISTENER-PORT=12020 -
X/W ,T-PROT=RFC1006 -
X/W ,TSEL-FORMAT=T -
X/W . . . .
```

X/W **PTYPE=UPIC-R connection**

X/W In the following example, port number 10030 is used in the local application and port number
X/W 12030 is used in the remote application. The remote application is running on the computer
X/W named CENTRAL4.

```
X/W BCAMAPPL BSPUPR -
X/W ,LISTENER-PORT=10030 -
X/W ,T-PROT=RFC1006 -
X/W ,T-SEL-FORMAT=T -
X/W . . . .
X/W
X/W PTERM UPRPART -
X/W ,PTYPE=UPIC-R,
X/W ,BCAMAPPL=BSPUPR -
X/W ,PRONAM=CENTRAL4 -
X/W ,LISTENER-PORT=12030 -
X/W ,T-PROT=RFC1006 -
X/W ,TSEL-FORMAT=T -
X/W . . . .
```


X/W In the KDCDEF call consistency checks are performed on the address information. The
X/W checks are performed because the OPTION statement uses the operand CHECK-
X/W RFC1006=YES

4.4.2 Providing address information with TNS entries

X/W In addition to using UTM generation to provide the address information for the transport
X/W system, you can also provide address information using TNS entries. The address infor-
X/W mation used when the application is run is controlled by the OPTION statement setting
X/W when KDCDEF is started. If the operands CHECKTNS=NO and CHECK-RFC1006=YES
X/W are specified for the KDCDEF run (default setting), then only the address information from
X/W the UTM generation is used.

X/W Otherwise openUTM will first attempt to find the address information in the TNS directory.
X/W If an appropriate TNS entry is found, then this entry is used with the highest priority. In this
X/W case the address information in the UTM generation is ignored.

X/W However, if no TNS entry is found, then the address information from the UTM generation
X/W is used.

X/W  It is recommended to always store all address information with the UTM generation.

X/W In order to administer the complete set of generation information of a UTM appli-
X/W cation at a common central location, you should replace existing TNS entries by
X/W storing the address information during UTM generation. You must then make sure
X/W that the TNS entries are deleted from the TNS directory.

X/W When the "mapped host name" function is used, the address information from the
X/W UTM generation is always used (see [section "Using mapped host names \(Unix
X/W systems and Window systems\)" on page 122](#)).

4.4.3 Converting address information from TNS entries to KDCDEF

X/W If you have been providing the address information to date using TNS entries, then you have
 X/W only entered in the UTM generation the application name, possibly the host name of the
 X/W communication partner, and the name of your UTM application. You must specify these
 X/W names in TNS as GLOBAL NAMES. The host name/IP address associations are deter-
 X/W mined by the TNS entry.

X/W When converting from TNS entries to KDCDEF, the only information you still need to specify
 X/W is the port number.

X/W Based on the LU6.1 connection from the [section “Providing address information with
 X/W KDCDEF”](#) (see [page 110](#)), the following presents an example of the changes you will need
 X/W to make when converting:

X/W LU6.1 connection

X/W In the following example, port number 10010 is used in the local application BSPLU61 and
 X/W port number 12010 is used in the remote application LU61PART. The remote application is
 X/W running on the computer named CENTRAL2.

X/W *Before conversion*

X/W KDCDEF: BCAMAPPL BSPLU61-
 X/W CON LU61PART -
 X/W ,BCAMAPPL=BSPLU61 -
 X/W ,PRONAM=CENTRAL2
 X/W , . . .

X/W TNS entries: BSPLU61\
 X/W TSEL RFC1006 T'BSPLU61'
 X/W TSEL LANINET A'10010'
 X/W
 X/W LU61PART.CENTRAL2\
 X/W TA RFC1006 *address of CENTRAL2* PORT 12010 T'LU61PART'

X/W *After conversion*

X/W When converting you must specify

- X/W ● for the local application name
- X/W the port number in the LISTENER-PORT parameter of the KDCDEF statement
- X/W BCAMAPPL instead of the port number in the TNS entry (TSEL LANINET).
- X/W ● for the remote partner
- X/W the processor name of the partner computer in the PRONAM parameter and the port
- X/W number in the LISTENER-PORT parameter of the KDCDEF statement CON instead of
- X/W the address and port number in the transport address of the TNS entry (TA).

X/W In order to be able to determine the IP address of the partner computer, the name CENTRAL2
X/W must be known to the DNS.

```
X/W KDCDEF:          BCAMAPPL BSPLU61 -
X/W                  ,LISTENER-PORT=10010 -
X/W                  ,T-SEL-FORMAT=T -
X/W                  . . . .
X/W
X/W                  CON LU61PART -
X/W                  ,BCAMAPPL=BSPLU61 -
X/W                  ,PRONAM=CENTRAL2 -
X/W                  ,LISTENER-PORT=12010 -
X/W                  ,TSEL-FORMAT=T -
X/W                  . . . .
```

X/W LU6.1 connection using symbolic names

X/W In the following example, port number 10010 is used in the local application BSPLU61 and
 X/W port number 12010 is used in the remote application LU61PART. The remote application is
 X/W running on the computer named CENTRAL2.

X/W Before conversion

X/W KDCDEF: BCAMAPPL LU61-
 X/W CON LU61 -
 X/W ,BCAMAPPL=LU61 -
 X/W ,PRONAM=PROLU61
 X/W , . . .

X/W TNS entries: LU61\
 X/W TSEL RFC1006 T'BSPLU61'
 X/W TSEL LANINET A'10010'
 X/W
 X/W LU61.PROLU61\
 X/W TA RFC1006 *address of CENTRAL2* PORT 12010 T'LU61PART'

X/W After conversion

X/W In contrast to the example [“LU6.1 connection” on page 114](#), you must also change the
 X/W BCAMAPPL name and the CON name in the TSEL parameters of the TNS entries. In this
 X/W case this means you must use the real names instead of the symbolic names.

X/W When converting you must

- X/W ● change the name of the local application
 X/W in the BCAMAPPL parameter to the value you used in the local TNS entry (TSEL
 X/W RFC1006).
- X/W ● for the local application name
 X/W as in the example [“LU6.1 connection” on page 114](#), specify the port number in the
 X/W LISTENER-PORT parameter of the KDCDEF statement BCAMAPPL instead of the port
 X/W number in the TNS entry (TSEL LANINET).
- X/W ● the name of the remote application
 X/W change the value in the CON statement to the value you used in the remote TNS entry
 X/W (TA).
- X/W ● for the remote partner
 X/W as in the example [“LU6.1 connection” on page 114](#), the processor name of the partner
 X/W computer in the PRONAM parameter and the port number in the LISTENER-PORT
 X/W parameter of the KDCDEF statement CON instead of the address and port number in
 X/W the transport address of the TNS entry (TA)

X/W In order to be able to determine the IP address of the partner computer, the name CENTRAL2
X/W must be known to the DNS.

```
X/W KDCDEF:          BCAMAPPL BSPLU61 -  
X/W                ,LISTENER-PORT=10010 -  
X/W                ,T-SEL-FORMAT=T -  
X/W                ,...  
X/W  
X/W                CON LU61PART -  
X/W                ,BCAMAPPL=BSPLU61 -  
X/W                ,PRONAM=CENTRAL2 -  
X/W                ,LISTENER-PORT=12010 -  
X/W                ,TSEL-FORMAT=T -  
X/W                ,...
```

4.5 Providing address information for the SOCKET transport system (Unix systems and Windows systems)

X/W In connections to a TS application with PTYPE=SOCKET, communication is performed via the socket interface using native TCP/IP as the transport protocol. For these socket clients the address information can only be provided with KDCDEF. openUTM obtains the IP addresses of the partner computer when started from the corresponding database, e.g. from the hosts file or the domain name server.

X/W The address information is stored in the PRONAM and LISTENER-PORT operands as well as in T-PROT and TSEL-FORMAT. In addition, the BCAMAPPL operand is also important.

X/W Note the following points in this context:

- X/W ● For PRONAM you must always specify the first TCP/IP host name generated in the database (hosts file, DNS, ...) or an assigned mapped host name (see [section “Using mapped host names \(Unix systems and Window systems\)” on page 122](#)). openUTM determines the IP address from it. Other names specified in the database (hosts file, DNS,...), e.g. aliases, must not be specified here.
- X/W ● You must always enter a port number for LISTENER-PORT. The port number absolutely must match the port number used by the communication partner. Specification of a LISTENER-PORT is mandatory.
- X/W ● You must always specify SOCKET for T-PROT.
- X/W ● It is recommended to set a value in the TSEL-FORMAT operand.
- X/W ● In BCAMAPPL you must specify an application name for which T-PROT=SOCKET was generated.

X/W PTYPE=SOCKET connection


X/W In the following example, port number 10010 is used in the local application and port number
X/W 12100 is used in the remote application LU61PART. The remote application is running on the
X/W computer named CENTRAL5.

X/W BCAMAPPL BPSOC -
X/W ,LISTENER-PORT=10100 -
X/W ,T-PROT=SOCKET -
X/W ,T-SEL-FORMAT=T -
X/W

X/W PTERM SOCPART -
X/W ,PTYPE=SOCKET ,
X/W ,BCAMAPPL=BPSOC -
X/W ,PRONAM=CENTRAL5 -
X/W ,LISTENER-PORT=12100 -
X/W ,T-PROT=SOCKET -
X/W ,TSEL-FORMAT=T -
X/W

4.6 Single- and multi-threaded network access (Unix systems and Windows systems)

X/W During distributed processing, the UTM application is connected to the network via network processes. These processes are responsible for handling connection setup requests and for managing data transfer via the connection. During generation, you can define whether the UTM application is to operate on the basis of single-threaded network processes or distributed over multi-threaded network processes.

X/W  For performance reasons, it is recommended to operate the UTM application with multi-threaded net processes. This is also the default setting.

X/W Use of multi-threaded and single-threaded net processes

X/W For distributed processing via CMX connections on Unix systems, you can use multi-threaded as well as single-threaded net processes.

X/W For distributed processing via CMX connections on Windows systems, you can only use multi-threaded net processes.


X/W For distributed processing via Socket connections, you can only use multi-threaded net processes.

X/W Multi-threaded network process with CMX and socket connections

X/W Multi-threaded network access allows you to maximize the process resources, since each network process is capable of managing several connections.

X/W The main process *utmmain* of a UTM application starts one or more network processes, which can in turn establish and manage numerous connections. Assignments between connections and processes are controlled through the use of **listener IDs**. All connections with the same listener ID are managed by threads of the same network process.

X/W There are different types of multi-threaded network process for CMX and socket connections. CMX connections use a process type called *utmnet*, and socket connections use *utmnets*.

X/W  Multi-threaded network access is defined in the **MAX statement** using the following operand:

- X/W ● NET-ACCESS=MULTI-THREADED


X/W Several parallel communication relationships are served by a single network process (default setting).

X/W  Listener IDs can be defined in the **ACCESS-POINT statement** for access points and in the **BCAMAPPL statement** for application names:

- X/W ● LISTENER-ID=number
- X/W This assigns a listener ID to the access point or application name as administrative information used for assigning processes during multi-threaded network access.
- X/W Due to the different types of multi-threaded net processes, the listener IDs for CMX connections and the listener IDs for Socket connections are separate value ranges.


X/W The connection is assigned to a network process by means of the listener ID allocated to the local application name or the local access point. If you have activated multi-threaded network access for an application, openUTM assigns the listener ID 0 to all application names and access points for which a listener ID has not been explicitly defined. All of these connections are then served by a single network process.

X Single-threaded network processes (for CMX connections on Unix systems only)

X  When assigning connections to network processes, you should take account of the **maximum** number of connections that the CMX transport system is able to support for each network process. If you do not, the partner application may provoke unwanted disconnections.


X The main process *utmmain* of a UTM application starts a net main process *utmmnetm*. The job of this process is to completely establish all CMX connections of the UTM application. The establishment of the connection can be initiated by the local UTM application as well as by a remote partner application.

X Once a connection has been established between the partners, *utmmnetm* starts a child network process *utmmnetc* for this connection. This manages the transfer of data between the partners, and continues to exist until the connection is shut down. A *utmmnetc* process exists for each connection established between the UTM application and its partner applications. With single-threaded network access, therefore, each connection is managed by a separate process.

X  Single-threaded network access is defined in the **MAX statement** using the following operand:

- X ● NET-ACCESS=SINGLE-THREADED


X Each communication relationship is administered by a separate network process.

X  For single-threaded network connections, the value of the listener ID you assigned in the access point or application name has no meaning.

4.7 Using mapped host names (Unix systems and Window systems)

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. The UTM host name is generally the same as the IP host name (real host name). As an option, you can also map a UTM host name to an IP host name. It makes sense to use host name mapping when


- X/W ● there are real host names longer than 8 characters.
- X/W ● the host name of a partner computer for the active establishment of a connection is different from the host name returned during the passive establishment of a connection (e.g. when a firewall is used).
- X/W ● the various host names in a cluster in the openUTM partner application are mapped internally to a fixed logical processor name of the cluster.

X/W  In this case, openUTM only maps the host names if a valid conversion file exists and if the UTM_NET_HOSTNAME environment variable is set (see the sections “[Conversion file for mapped host names](#)” on page 123 and “[UTM_NET_HOSTNAME environment variable](#)” on page 125).

X/W Generation

X/W If you want to use host name mapping, proceed as follows:

- X/W ► Create a conversion file and specify the mappings of the UTM host names to the real host names (see [page 123](#)).
- X/W ► Set the UTM_NET_HOSTNAME environment variable to activate the conversion file (see [page 125](#)).
- X/W ► Change your input for the generation so that you specify a UTM host name and not the real host name as the value for the host name in the corresponding KDCDEF statements.

X/W  If the UTM_NET_HOSTNAME environment variable is set to a valid conversion file, then TNS functionality is deactivated for the entire UTM application.

X/W Runtime

X/W openUTM converts a UTM host name to a real host name and vice-versa at runtime:

X/W ● If there is a valid conversion file available and a connection is established actively, then an attempt is made to convert the UTM host name of the partner computer to a real host name before openUTM determines the associated IP address.

X/W If it is impossible to convert the host name, then the IP address is determined from the UTM host name.

X/W ● When connection is established passively, then the real host name is determined from the IP address of the partner. If there is a valid conversion file available, then an attempt is made to convert the real host name to a UTM host name.

X/W – If the conversion was successful, then openUTM compares the generation information to the converted host name.

X/W – If the conversion was unsuccessful, then openUTM compares the generation information to the real host name.

X/W **i** You can check if the UTM host name was converted to a real host name or vice-versa for a connection based on the value of the processor name in the `TNSNAME` insert in the messages from the net process. If a conversion was performed, then the value for the processor name consists of the UTM host name in parentheses together with the real host name, e.g. `system1(central)`. If no conversion was performed, then the value for the processor name consists of just the real host name, e.g. `central`.

4.7.1 Conversion file for mapped host names

X/W The conversion file contains all rules according to which UTM host names are converted to real host names and vice versa.

X/W openUTM reads the conversion file when a UTM application is started and generates an internal host name table from the data. The information in this table is used at runtime to convert the host names.

X/W During the installation of openUTM, a sample file named `utmhostname` is installed. You will find the file directly in the `utmpfad` directory of openUTM.

X/W **i** The conversion file does not need to contain an entry for every host name generated, but only for the host names for which mapping is performed. The IP address is determined from the UTM host name for all generated host names for which there are no conversion rules.

X/W File name

X/W You can choose any name you wish for the conversion file.

X/W Default: *utmhostname*

X/W The filename may be a maximum of 300 characters long.

X/W File format

X/W The conversion file is a line-based file. The lines have the following format:

X/W # (#) openUTM Host Name File

X/W

X/W # Conversion rules

X/W

X/W *utm_hostname_1_real_hostname_1[real_hostname_2]...*

X/W *....*

X/W *utm_hostname_n_real_hostname_m[real_hostname_o]...*

X/W *....*

X/W # (#) openUTM Host Name File

X/W Header of the conversion file. The header must always be placed in the first line of the file.

X/W Empty line

X/W Empty lines are ignored when the file is read.

X/W *#Comments*

X/W Comment lines begin with the '#' character in column 1 and are ignored when the file is read.

X/W *utm_hostname_x*

X/W UTM host name for which conversion to a real host name and vice versa is to be performed.

X/W The UTM host name must start in column 1.

X/W Maximum length: 8 characters

- X/W *real_hostname_y*
 X/W Real host name of a partner computer as it is entered in the name service of the
 X/W local system (e.g. in the hosts file). You can specify several real host names
 X/W separated by one or more space/tab characters in a conversion rule.
- X/W If you have specified a real host name here for which no IP address can be found,
 X/W then you will receive a K154 message (IP address could not be found) during
 X/W runtime of the UTM application.
- X/W Maximum length: 64 characters
- X/W The following maximum values and rules apply to the conversion file:
- X/W ● Maximum number of conversion rules per file: 500
 - X/W ● A conversion rule ends at the end of the line, there are no continuation lines.
 - X/W ● Maximum line length: 300 characters
 - X/W ● Maximum number of real host names in the conversion file: 2000
 - X/W ● Several conversion rules can be present for a single UTM host name.
 - X/W ● The order of the conversion rules in the file determines the priority of the conversion of
 X/W host names at runtime.
 - X/W ● Empty lines and comment lines are ignored.
 - X/W ● Lines containing syntax errors are ignored.

4.7.2 UTM_NET_HOSTNAME environment variable

- X/W The UTM_NET_HOSTNAME environment variable specifies the conversion file for host
 X/W names that will be assigned to the UTM application (see openUTM manual "Using
 X/W openUTM Applications under Unix systems and Windows systems" under "General
 X/W environment variables").
- X/W If this variable is set, then the conversion file specified is evaluated when a UTM application
 X/W is started.
- X/W The environment variable contains the full file name of the conversion file including the path.
 X/W If you only set UTM_NET_HOSTNAME and do not specify a file name, then openUTM
 X/W searches for the *utmhostname* file in the local directory (the directory in which the utmain
 X/W process was started).

5 Generating selected objects and functions of the application

This chapter describes how to configure certain objects of your UTM application and explains which KDCDEF control statements or which of the individual operands are relevant for describing the objects. This applies to the following UTM objects:

- Clients ([page 128](#))
- B/X ● Printers, printer control LTERMs, printer pools under BS2000/OSD and Unix systems ([page 168](#)) as well as RSO printers connected to a UTM application under BS2000/OSD ([page 172](#))
- B/X ● Service-controlled queues ([page 181](#))
- Message modules ([page 186](#))
- B ● Multiplex connections of a UTM application under BS2000/OSD ([page 192](#))
- B ● BLS load modules and common memory pools of a UTM application under BS2000/OSD ([page 199](#))

In addition, several selected UTM functions are described here that affect KDCDEF control functions and whose operands are significant to the use of these functions. This is true for the following functions:

- Job control using priorities and process constraints ([page 208](#))
- Access control functions ([page 219](#))
- Encryption of messages on connections to clients ([page 228](#))
- Coupling of resource managers and databases ([page 234](#))
- B ● Internationalization of a UTM application under BS2000/OSD ([page 237](#))
- B ● System access control using Kerberos under BS2000/OSD ([page 243](#))

See also the openUTM manual “Concepts und Functions”.

5.1 Connecting clients to the application

This section describes the generation of terminals, UPIC clients, transport system applications and OpenCPIC clients. Transport system applications are DCAM, PDN, CMX and socket applications as well as UTM applications that are generated as transport system applications. These will subsequently be referred to as TS applications.

The options that the UPIC clients offer are described in detail in the manual “openUTM-Client for the UPIC Carrier System”.

Each client that wants to use the services of a UTM application must be known to the UTM application. A client is known to a UTM application if it is assigned to a logical connection point defined in the configuration. There are various different types of client:

- For terminals, UPIC clients and TS applications, a logical connection point is known as an **LTERM partner**. There are two methods of connecting to an LTERM partner:
 - You generate the client for an individual connection by defining the physical client using a PTERM statement and then assigning an exclusive LTERM partner, see below. A client must always be generated with PTERM, when connections to this client are to be established in the UTM application (e.g. to TS applications). You only need to issue a PTERM statement to other clients when you want to assign the other client a specific logical property, e.g. special access rights that you do not want to assign to any LTERM pool.
 - You define a pool of LTERM partners, also called an LTERM pool, see [page 133](#). You can connect several clients using the LTERM pool.
- For OpenCPIC clients, the logical connection point is known as the **OSI LPAP partner**. It is possible to establish several parallel connections via an OSI LPAP partner.

The first two sections show the basic steps required to connect a client. The [section “Defining the client sign-on services” on page 144](#) through [section “Examples of the generation of a client/server cluster” on page 163](#) go into more detail on certain topics, including the signing-on process, security functions and addressing.

5.1.1 Connecting clients via LTERM partners

If you wish to connect physical terminals, UPIC clients and TS applications individually, you will need to supply the following generation statements for each client:

- an LTERM statement for the logical connection point
- a PTERM statement for the physical client.

UPIC clients and TS applications may also require a BCAMAPPL statement. Limits, maximum values and parameters, which are to be set throughout the application for communication between clients and the UTM application, are defined in the MAX statement.

LTERMs and PTERMs may also be created dynamically (objects KC_LTERM and KC_PTERM). Moreover, the assignment of the client to the LTERM partner in the PTERM statement can be adapted dynamically at a later stage using administration functions. For example, you can assign another client (of the same type) to an LTERM partner during operation, or assign another LTERM partner – for which you may have defined different access rights – to a client. See also the openUTM manual “Administering Applications”.



LTERM statement on [page 355](#)

The most important properties for LTERM partners, via which clients can connect to an application, are defined with the following operands:

- *ltermname*
Name of the LTERM partner. Logical name via which the client, to which the LTERM partner is assigned, is addressed by the program units of the application.
- KSET=
Key set of the LTERM partner, i.e. an authorization profile that defines which parts of the application program (which TACs) are available to the client connecting to the application via this LTERM partner.
- LOCALE= (only BS2000/OSD)
LTERM-specific language environment of the clients that connect to the application via this LTERM partner. This language environment is also used by openUTM to output messages, as long as no user is signed on.
- LOCK=
Lock code as system access control. The connection is only established when the client signs on to openUTM with a user ID, for which a key set was generated, using a key code corresponding to this lock code.

B

B

B

B

- **USAGE=D**
Type of communication partner. In this case, dialog partners are connecting to the application via the LTERM partner. Messages can be exchanged in both directions.
- **USER=**
The user ID under which the client is automatically signed on when a connection has been established, see [section “Automatic sign-on under a specific user ID” on page 145](#). You are also able to define other characteristics for this user ID, see [section “Generating security functions” on page 148](#).



PTERM statement on [page 437](#)

The most important properties for physical clients are defined with the following operands:

- *ptermname*
Name of the client as generated in the system of the server application.

B
B

BS2000/OSD (without Socket application):

The BCAM name of the client must be specified.

Socket applications:

If the connection is to be established from the local UTM application to the client, then any *ptermname* can be selected. Otherwise, the name must have the format `PRTnnnnn`. Here *nnnnn* means the port number used by the socket application to establish the connection.


See [section “Providing address information” on page 153](#) for more information.


- **BCAMAPPL=**
Name of the local application via which the transport system establishes the connection between the client and the UTM application. This name must be defined in a BCAMAPPL statement or using `MAX ...APPLINAME=`. If you omit this operand, the name is taken from `MAX ...APPLINAME=`. Terminals may only use names that are defined in `MAX ... APPLINAME=`.
- **ENCRYPTION-LEVEL=**
For UPIC clients and under BS2000/OSD additionally for certain terminal emulations you specify the minimum encryption level that must be maintained on the connection to the client. You can specify `trustworthy` for the client, which means that this client is permitted to work with the UTM application without encryption. See also [section “Message encryption on connections to clients” on page 228](#) for more information on encryption.



- LTERM=

The LTERM partner *ltermname*, via which the client connects to the UTM application, is assigned to the physical client as a logical connection point.
- PRONAM=

Symbolic name of the processor on which the client resides.
- PTYPE=

Type of client connected via the LTERM partner. Here you specify whether the client is a transport system application, a UPIC client or a terminal.
-  ● T-PROT=, TSEL-FORMAT= (only under Unix systems and Windows), LISTENER-PORT= (with PTYPE=SOCKET also under BS2000/OSD)

Components of the transport address of a remote UPIC client or a TS application see [section “Providing address information” on page 153](#).
-  ● USAGE=D (only BS2000/OSD)

  USAGE=D defines that the communication partner is a dialog partner. Messages can be exchanged between the UTM application and the client.
- USP-HDR=

With a sockets application, this parameter controls which of the output messages openUTM is to create a protocol header for, see [section “USP headers for output messages to socket connections” on page 151](#).



BCAMAPPL statement on [page 291](#)

It is possible to define additional application names for UPIC clients and TS applications.

- *appliname*

Name of the local application used by the transport system to establish the connection between the client and the UTM application. If this name is used for socket applications, it may not be used by a different type of partner.
- SIGNON-TAC=

Specifies if and when a sign-on service takes place, when a client attempts to sign on under this application name, see [section “Generating sign-on services for clients” on page 145](#).

X/W

- TSEL-FORMAT=, LISTENER-ID= (only under Unix systems and Windows), LISTENER-PORT= (with PTYPE=SOCKET also under BS2000/OSD) T-PROT=

For information about the components of the transport address under which client contacts the UTM application, see [section “USP headers for output messages to socket connections” on page 151](#).



MAX statement on [page 368](#)

Default and maximum values, which are relevant for communication of clients with the UTM application, are defined with the following operands:

- CONN-USERS=

Controls the utilization of the application. The operand defines the maximum number of users who can simultaneously work with the application. In the case of an application for which no user IDs are generated, CONN-USERS= defines the maximum number of clients that can simultaneously connect to the application via LTERM partners.

- TRMSGGLTH=

Maximum length of the physical messages that can be exchanged between clients and the UTM application.

B

- LOCALE= (under BS2000/OSD only)

B

B

B

B

B

B

Defines the default language environment (locale) of the UTM application. The locale generated here is assigned to the clients connected via LTERM partners or LTERM- pools as the default value for the language environment. The default setting applies unless a specific locale is defined for these objects in the corresponding LTERM or TPOOL statements. See also [section “Defining the language environment – setting the locale” on page 240](#).

5.1.2 LTERM pools

A particular number of LTERM partners with the same logical properties are defined for an LTERM pool as logical connection points for clients. Different clients with the same technical properties (partner and processor type) can connect dynamically to a UTM application via these LTERM partners. The assignment only applies for the duration of a session; there is no static assignment between a client and an LTERM partner.

An LTERM pool must be configured in a TPOOL statement (in place of LTERM/PTERM statements). In addition, in the same way as for a single connection, a BCAMAPPL statement may be necessary, see [page 131](#). The settings in the MAX statement are also valid for LTERM pools, see [page 132](#).

Various types of LTERM pools can be configured:

- LTERM pools via which only clients of a particular type (PTYPE=), located on a particular processor (PRONAM=), can connect to a UTM application.
- LTERM pools, via which clients of a particular type can connect to a UTM application, regardless of the processor on which they reside (open LTERM pools).

B
B In UTM applications under BS2000/OSD you can also generate the following types of LTERM pools:

B
B ● LTERM pools for all clients with a user services protocol, regardless of type, yet located on a particular processor.

B
B ● LTERM pools for all clients with a user services protocol, regardless of type and the computer on which it is located.



TPOOL statement on [page 511](#)

The most important properties for LTERM pools are defined with the following operands:

- BCAMAPPL=

Name of the local application via which the transport system establishes the connection between the client and the UTM application. The name must be defined in a BCAMAPPL statement or using MAX ...APPLINAME=. If you omit this operand the name is taken from MAX ...APPLINAME=.

B
B In BS2000, clients with the user services protocol may only use the name from MAX ... APPLINAME=.

- **CONNECT-MODE=**

With **CONNECT-MODE=** you specify if a UPIC client or TS application may connect to the application multiple times under the same name via the LTERM pool.
- **KSET=**

Key set of the LTERM pool that uses key codes to define the access rights of the clients which connect to the UTM application via the LTERM pool.
- **USER-KSET=**

In UTM applications with user IDs the **USER-KSET** key set for UPIC clients and TS applications specify limited system access rights (in comparison with **KSET**). The key set in **USER-KSET** takes effect when the client does not pass a user ID to `openUTM` while establishing the connection/conversation or while in the sign-on service.
- **LOCK=**

System access control of the LTERM pool, i.e. lock code assigned for all LTERM partners of the pool. The connection is only established if the client signs on to `openUTM` with a user ID whose key set has the corresponding key code.
- **ENCRYPTION-LEVEL=**

For UPIC clients and under BS2000/OSD additionally for certain terminal emulations you specify the minimum encryption level that must be maintained on the connection to the client. You can specify trustworthy for the client. See also [section “Message encryption on connections to clients” on page 228](#) for more information on encryption.
- **LTERM=**

LTERM prefix from which unique LTERM partner names are created with *number* LTERM partners of the LTERM pool.
- **NUMBER=**

Number of LTERM partners configured for this LTERM pool. This also implicitly defines the maximum number of clients that can connect to this LTERM pool.
- **PRONAM=**

Name of processor which must contain the client connected via the LTERM pool.
- **PROTOCOL=**

Specifies if the user services protocol is used or not.

B

B

- PTYPE=
Type of client connected via the LTERM pool.

B

- LOCALE=

B
B
B

LTERM-specific language environment that applies to all clients which connect to the application via LTERM partners. This language environment is also used by openUTM to output messages, as long as no user is signed on.

Assignment of client when connecting via an LTERM pool

For clients that want to connect to an application via an LTERM pool, please note that openUTM only assigns a client to one LTERM pool or no LTERM pool. When selecting the LTERM pool, openUTM considers it more important to match the processor name than the client type.

The table below shows the sequence in which openUTM attempts to connect a client using the generated PTERMs and LTERM pools. The shaded rows in the table represent LTERM pools that can only exist in a UTM application under BS2000/OSD. These are ignored under Unix systems and Windows.

B
B

B
B

Assignment of client		KDCDEF statements: definition of client		
1	↓	PTERM	LTERM=ltermname PRONAM=processorname	
2		TPOOL	PTYPE=partnertype PRONAM=processorname	
3		TPOOL	PTYPE=*ANY PRONAM=processorname	<i>only in BS2000/OSD</i>
4		TPOOL	PTYPE=partnertype PRONAM=*ANY	
5		TPOOL	PTYPE=*ANY PRONAM=*ANY	<i>only in BS2000/OSD</i>

1. When establishing a connection, it is first of all checked whether a PTERM statement exists for the client. A client that was explicitly generated with a PTERM statement cannot connect to a UTM application via an LTERM pool.
2. If an LTERM pool is generated for the processor name (PRONAM) and type (PTYPE) of a client, this client is assigned to this LTERM pool or to no LTERM pool.
3. (only in BS2000/OSD)
If there is no LTERM pool with the processor name and type of the client, the client is assigned to the LTERM pool with the same processor name and PTYPE=*ANY.
4. If no such LTERM pool exists, the client is assigned to an “open” LTERM pool with the same type and PRONAM=*ANY, i.e. all clients of a type can connect to the UTM application, regardless of the processor on which they reside.
5. (only in BS2000/OSD)
If no such LTERM pool exists either, the client is assigned to an LTERM pool with PTYPE=*ANY and PRONAM=*ANY.

If there is no LTERM pool of this type, the connection setup request is rejected.

5.1.3 LTERM bundle

With a LTERM bundle (connection bundle) you distribute queued messages to a logical partner application equally among several parallel connections. The logical partner application can comprise several instances of the partner application (e.g. a UTM cluster application). This type of procedure makes sense when a UTM application sends a very large number of queued messages to a partner application, possibly leading to the overloading of the transport connection.

You define a LTERM bundle using LTERM and PTERM statements as already described in the [section “Connecting clients via LTERM partners” on page 129](#). The following text describes the additional points you must note to work with LTERM bundles.

A LTERM bundle consists of one master LTERM and several slave LTERMs. The slave LTERMs, which must be assigned using PTERM with PTYPE=APPLI or PTYPE=SOCKET, are assigned to a master LTERM through generation.

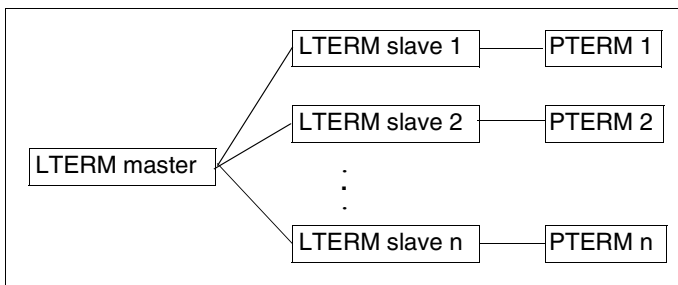


Figure 11: Example of a LTERM bundle

FPUT/DPUT calls

FPUT and DPUT calls sent by program units to the master LTERM are assigned to one of the slave LTERMs at the end of the transaction:

- openUTM first attempts to find a slave LTERM whose PTERM is connected. If openUTM cannot find such a connection, then it searches for a slave LTERM that was generated with RESTART=YES.
If openUTM finds a slave LTERM, then all queued messages sent in this transaction to this master LTERM are assigned to the slave LTERM.
- If openUTM cannot find a slave LTERM, then all messages sent to the master LTERM with FPUT or DPUT are rejected.
- If a slave LTERM is generated with RESTART=NO and the connection is cleared or lost, then all messages pending output on this LTERM are rejected.

Program units can also send FPUT and DPUT calls directly to the slave LTERMs. However, these FPUTs are not subject to the distribution algorithm described above.

Information displayed in the KB header

Messages can also be received through the slave LTERMs of a LTERM bundle. In services started for received messages, openUTM always displays the name of the LTERM through which the message was received in the KB header. The following therefore applies for LPAP bundles:

In services started for messages received through a slave LTERM, the name of this slave LTERM is displayed in the KB header and **not** the name of the master LTERM.

With the aid of the KDCS call INIT PU you can obtain information on whether the LTERM in the KB header is the slave of an LTERM bundle as well as the name of the master LTERM (see the openUTM manual “Programming Applications with KDCS”).



LTERM statement on [page 355](#)

In addition to the properties for LTERM partners already listed (see [page 129](#)), the following operands must also be specified for LTERM bundles:

- BUNDLE=

Specifies the corresponding master LTERM in the definition of a slave LTERM. The master LTERM specified here must have been generated in a preceding LTERM statement:

```
LTERM master, ...
LTERM slave1, BUNDLE=master, ...
LTERM slave2, BUNDLE=master, ...
```

```
PTERM slave1, LTERM=slave1, PTYPE=APPLI|SOCKET, ...
PTERM slave2, LTERM=slave2, PTYPE=APPLI|SOCKET, ...
```

- RESTART=

Determines how queued messages are handled when the connection to the client is cleared. Messages pending output on a LTERM that were generated with RESTART=NO may be rejected if necessary (see the section “[FPUT/DPUT calls on page 137](#)”).



All LTERM parameters of slave LTERMs except for *ltermname*, USER, QAMSG, RESTART, and STATUS must match the same parameters of the master LTERM. Otherwise they will be overwritten by KDCDEF using the data specified in the master LTERM. No message is output in this case.

When assigning the FPUT and DPUT calls to a slave LTERM at the end of a transaction, the QAMSG and RESTART settings are evaluated on the slave LTERM.

All slave LTERMs in a LTERM bundle should be generated identically. KDCDEF does not check these specifications, though.



PTERM statement on [page 437](#)

In addition to specifying the properties for physical clients already listed (see [page 129](#)), the following operands must also be specified for the PTERMs assigned to the slave LTERMs in a LTERM bundle:

- PTYPE=APPLI | SOCKET

All PTERMs in a LTERM bundle must be generated with PTYPE=APPLI or PTYPE=SOCKET. The same PTYPE must be specified here for all PTERMs in a LTERM bundle.

B

- USAGE=D (BS2000/OSD only)

B

All PTERMs in a LTERM bundle must be generated with USAGE=D.



All PTERMs in a LTERM bundle should address the same partner application or a partner application of the same type. KDCDEF does not check these specifications, though.

5.1.4 LTERM groups

In a LTERM group you assign one or more LTERMs a connection. If a UTM application is to send queued messages to different partner applications depending on which functional area the message belongs to, then a separate LTERM must be assigned to each functional area in the partner application.

The program units direct their FPUT and DPUT calls to the appropriate LTERM depending on the function. If the partner application to function relationship is 1:1, then each LTERM is assigned one PTERM. If the partner application to function relationship is n:1 and the assignment may change in some cases, then n LTERMs are assigned to one PTERM.

An LTERM group consists of one or more alias LTERMs, called the group LTERMs, and one primary LTERM. You define the group LTERMs using LTERM statements as described in the [section “Connecting clients via LTERM partners” on page 129](#). Do not assign a PTERM to a group LTERM.

The primary LTERM must be a normal LTERM or the master LTERM of a LTERM bundle. If the primary LTERM is a normal LTERM, then a PTERM with PTYPE=APPLI or PTYPE=SOCKET must be assigned to it. You define the primary LTERM as described in [section “Connecting clients via LTERM partners” on page 129](#).

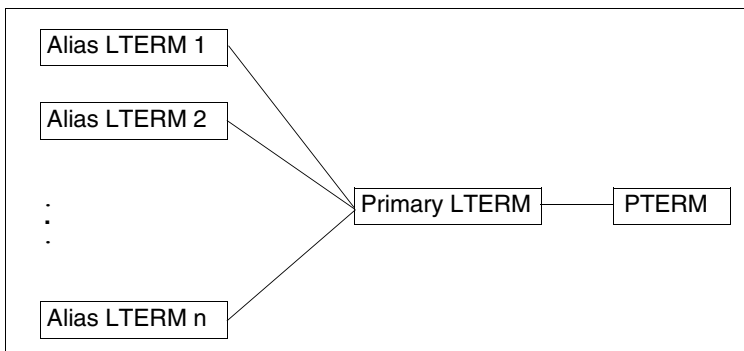


Figure 12: Example of a LTERM group

LTERM groups can also be used in conjunction with LTERM bundles. In this case the primary LTERM is the master LTERM of the LTERM bundle.

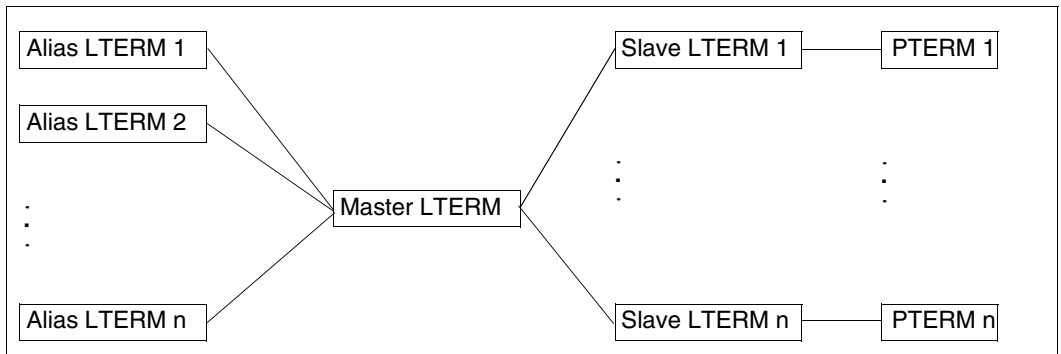


Figure 13: Example of a LTERM group in conjunction with a LTERM bundle

FPUT/DPUT calls

FPUT and DPUT calls sent by program units to an alias LTERM are processed as follows:

In a LTERM group without a LTERM bundle:

FPUT and DPUT calls sent to an alias LTERM are sent by openUTM via the PTERM assigned to the primary LTERM.

In a LTERM group whose primary LTERM is the master LTERM of a LTERM bundle:

If FPUT and DPUT calls are sent to an alias LTERM in this kind of LTERM group, then all queued messages sent in the transaction to alias LTERMs in the group are assigned by openUTM to exactly one of the slave LTERMs at the end of the transaction.

This procedure guarantees that the recipient receives the messages in the same order as they would be generated in a transaction for an LTERM group.

Program units can also send FPUT and DPUT calls directly to the primary LTERM.

Information displayed in the KB header

If the primary LTERM of a LTERM group is not the master LTERM of a LTERM bundle, then messages can also be received via the primary LTERM. In services started for received messages, openUTM always displays the name of the LTERM or LPAP from which the message was received in the KB header. The following also applies to LTERM groups: In services started for messages received via the primary LTERM, the name of the primary LTERM is displayed in the KB header and **not** the name of an alias LTERM.

With the help of the KDCS call INIT PU you can obtain information on whether the LTERM in the KB header is the primary LTERM of a LTERM group (see openUTM manual “Programming Applications with KDCS”).



LTERM statement on [page 355](#)

In addition to the properties for LTERM partners already listed (see [page 129](#)), the following operands must also be specified for a LTERM group:

- GROUP=

Specifies the corresponding primary LTERM in the definition of an alias LTERM. The primary LTERM specified here must have been generated in a preceding LTERM statement:

```
LTERM primary, ...
PTERM primary, LTERM=primary, PTYPE=APPLI | SOCKET, ...
```

```
LTERM alias1, GROUP=primary, ...
LTERM alias2, GROUP=primary, ...
```



All LTERM parameters of the alias LTERMs except for *ltermname*, USER, and STATUS must match the same parameters of the primary LTERM. Otherwise they will be overwritten by KDCDEF using the data specified in the primary LTERM. No message is output in this case.

Only the generation parameters of the primary LTERM are evaluated for a FPUT or DPUT call.



PTERM statement on [page 437](#)

In addition to specifying the properties for physical clients already listed (see [page 129](#)), the following operands must also be specified for the PTERM assigned to the primary LTERM of a LTERM group:

- PTYPE=APPLI | SOCKET

The PTERM in a LTERM group must be generated with PTYPE=APPLI or PTYPE=SOCKET.

B

- USAGE=D (BS2000/OSD only)

B

The PTERM in a LTERM group must be generated with USAGE=D.

5.1.5 Connecting OpenCPIC clients

OpenCPIC clients are treated as OSI TP partners. For this reason, this section only describes the client-specific features of OSI TP generation.

Generation

An OpenCPIC client is generated in a similar way to a server/server link, see [section "Distributed processing via the OSI TP protocol" on page 93](#). The only difference is that LTAC statements are not required if it is just a client.

The statements ABSTRACT-SYNTAX, APPLICATION-CONTEXT and TRANSFER-SYNTAX are only necessary if you want to define your own application context.

More information about addressing and coordination with the OpenCPIC generation can be found as of [page 160](#).

5.1.6 Defining the client sign-on services

This section describes the interface between generation and the sign-on service for clients if the application is generated using user IDs. The sign-on service is made up of the following two stages: **establishing the connection** and **signing on**.

The connection is established using the application names as specified in the operand BCAMAPPL= or, for OpenCPIC, specified in the operand LOCAL-ACCESS-POINT=.

Signing on to a UTM application

Signing on to a UTM application is carried out using a user ID. The following stages are required, regardless of whether the default sign-on service is used or another sign-on service:

- When using terminals, the terminal user must prove their authorization once a connection has been established. To do this the user must enter at least one user ID. This user ID must be generated in a USER statement. This is also called the **real user ID**.
- TS applications and UPIC clients are signed on after connection using a so-called **connection user ID**. This is a user ID which is implicitly generated by openUTM using the LTERM name if no user ID is specified in the operand USER= of the LTERM statement. If a user ID is specified in the operand USER= (explicit connection user ID), then this must be generated with a USER statement, see [section “Automatic sign-on under a specific user ID” on page 145](#). This user ID cannot be used as a real user ID.
- OpenCPIC clients are signed on under their **association names** once the association has been established. The association names are formed using the names specified in the operand ASSOCIATION-NAMES= and a sequential number, for example, ASSOC03, see [page 426f](#).

UPIC clients, TS applications and OpenCPIC clients can then subsequently sign on using a real user ID.

The execution of the sign-on service can be defined by the generation, for example, using automatic connection establishment, automatic sign-on under a specific user ID, separate sign-on service or by permitting multiple sign-ons.



Detailed information about the sign-on service can be found in the relevant section of the openUTM manual “Using openUTM Applications”. The individual steps required for a client to sign on to a UTM application are described there.

5.1.6.1 Establishing an automatic connection

Certain clients can be generated in such a way that openUTM attempts to establish a connection to the client as soon as the application is started. This is possible when using:

- OpenCPIC clients,
- individually generated terminals and TS applications in BS2000/OSD
- individually generated TS applications in Unix systems and Windows.

An automatic connection can be established as follows:

- For TS applications and terminals:

```
PTERM ... ,CONNECT=YES
```

- For OpenCPIC clients:

```
OSI-LPAP ... ,CONNECT=n (n>0)
```

For OpenCPIC clients, the client must also be generated using PARTAPPL ... ,CONNECT=*n* (*n*>0), see the openUTM manual “Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications”.

5.1.6.2 Automatic sign-on under a specific user ID

You can explicitly assign all clients defined using LTERM/PTERM a user ID under which this client automatically signs on once a connection is established. If you do so, the authorizations that apply to that client are the ones assigned to the specified user ID, see [section “Generating security functions” on page 148](#). To do this you will need the following generation statements:

```
LTERM ... USER=username
USER username ...
```

Terminals are then always signed on under this user ID. For TS applications and UPIC clients this user ID is a connection user ID and thus cannot be replaced by a real user ID, e.g. in a sign-on service, see below.

5.1.6.3 Generating sign-on services for clients

It is possible to program special sign-on services for terminals, UPIC clients and TS applications. A sign-on service is linked to an application name. This means that you can assign a sign-on service to any application name. Application names are defined using MAX APPLNAME= or in a BCAMAPPL statement.

If a client signs on using a specific application name, then the sign-on service assigned to this application name is started. The application name under which the client signs on is specified in PTERM/TPOOL in the operand BCAMAPPL.

Sign-on services are generated as follows:

- The sign-on service for the default application name (as defined in MAX ... APPLNAME) is generated using:

```
TAC KDCSGNTC , PROGRAM=signon-prog1
PROGRAM signon-prog1 ...
```

signon-prog1 is the name of the program unit that is initially run in the sign-on service.

If a default application name is generated for a sign-on service, this is then taken as the default value for all application names generated using BCAMAPPL.

- The sign-on service for an application name defined using BCAMAPPL is generated using:

```
BCAMAPPL appliname2...,SIGNON=signon-tac
TAC signon-tac , PROGRAM=signon-prog2
PROGRAM signon-prog2
```

signon-prog2 is the name of the program unit that is initially run in the sign-on service.

- If sign-on services are also to be run for UPIC clients, you must specify the following in the SIGNON statement:

```
SIGNON ... UPIC=YES
```

If this setting is not made, UPIC clients cannot use sign-on services, not even if an appropriate sign-on service has been generated for the application name.

More information about the programming can be found in the openUTM manual "Programming Applications with KDCS".

5.1.6.4 Multiple sign-ons

When using user IDs, it is usually only possible for a single client to sign on to an application at any given time, a second attempt to sign on under the same user ID is thus rejected.

If you want to enable multiple sign-ons for specific user IDs you must generate the following:

```
SIGNON ... MULTI-SIGNON=YES
```

This means that it is possible, at any given time, for several clients to sign on to an application under a single real user ID without a restartable service context (USER *username*... RESTART=NO), but only **one** of these clients may be a terminal client.

OpenCPIC clients that have selected the functional unit "Commit", may perform a multiple sign-on under any real user ID.

5.1.7 Specifying maximum waiting times for dialog prompting

In the KDCDEF control statement MAX, you can specify the maximum waiting times for dialog prompting using the operand TERMWAIT= and PGWTIME= as well as the IDLETIME= operand of the PTERM statement.

- The operand PGWTTIME= is used to set the maximum permitted length of the interval between the output of a dialog message to the client after a blocking call (for example, a PGWT call) and the subsequent dialog input. If no input is made during this period of time, openUTM is forced to interrupt the service.
- The operand TERMWAIT= is used to set the maximum permitted length of the interval between the dialog output at a terminal after a PEND KP and the subsequent dialog input. If no input is made during this period of time, openUTM is forced to interrupt the service.
- The operand IDLETIME= is used to limit the waiting time after PEND RE and PEND FI/ER/FR, or in other words, after the end of a transaction. Monitoring the waiting time after the end of a transaction is used for data protection purposes. Should a user forget to sign off after completing work with the application, this function allows you to reduce the risk of unauthorized persons working on the client **without** signing on.

Remember that with non-blocking calls the UTM process is released and its able to take on other tasks, whereas with a blocking call the process remains occupied. If a service waits after a PEND KP call or PGWT KP call for a dialog input, then the transaction remains open (“multi-step transaction”) and usually keeps global resources locked (for example, GSSBs, data areas in database systems). A delay in the dialog input would block the work of other users wanting to access this data. Monitoring the time intervals for the instances indicated above can help to avoid this undesired effect.



The attention of the user must be drawn towards this fact. The users must be shown the critical points of the dialog interaction so that they are aware of the effect an input delay may have on the performance of the application as a whole.

5.1.8 Generating security functions

The security functions are made up of the following components:

- **System access control:**
System access control is defined in the USER statement, see below.
- **Administration authorization:**
Administration authorization is assigned in the USER statement or the OSI-LPAP statement (OpenCPIC), see [page 149](#).
- **Data access control:**
Data access control is specified using the operands KSET, USER-KSET or ASS-KSET of the USER, LTERM, TPOOL or OSI-LPAP statement. Data access protection must be defined within the framework of a lock/key code concept or of the access list concept and is described in detail in [section “Data access control” on page 219](#). Data access control for OpenCPIC clients is generated in the same way as described in section [“Protection measures in the receiving application” on page 227](#).
- **Encryption:**
The encryption level is specified in the operand ENCRYPTION-LEVEL of the PTERM, TPOOL or TAC statement. A detailed description of message encryption can be found in [section “Message encryption on connections to clients” on page 228f](#). Encryption by openUTM is not supported for OpenCPIC clients.

5.1.8.1 Defining system access control

System access control is only relevant for real user IDs and is generated in the USER statement.

You define system access control by assigning a password to each user ID and by specifying a certain level of complexity for the password. A hexadecimal password may not be specified for client/server communication.

```
USER userid-name ,PASS=password ,PROTECT-PW=complexity-level
```

On signing on, the client must pass the specified values for *userid-name* and *password* to openUTM.

5.1.8.2 Assigning administration authorizations

- You can specify the administration authorizations for the user ID using the USER statement:

```
USER userid-name,PERMIT=ADMIN
```

B
B
B

Under BS2000/OSD you can also assign UTM-SAT administration authorizations for a user ID (PERMIT operand) and specify the type and range of the SAT logging (SATSEL operand):

B

```
USER userid-name ,PERMIT=SATADM ,SATSEL=...
```

- You can assign administration authorization to an OpenCPIC client in OSI-LPAP:

```
OSI-LPAP ... PERMIT=ADMIN
```

B
B

Under BS2000/OSD you can also assign UTM-SAT administration authorizations for a client:

B

```
OSI-LPAP ... PERMIT=SATADM or PERMIT=(ADMIN,SATADM)
```

If the OpenCPIC client signs on under a real user ID, then the data access rights that are generated for that user ID are valid and not the data access rights of the OSI-LPAP.

5.1.9 Generating a restart

The restart function for a client is linked to the user ID that the client has used to sign on to the UTM application.

Restart function for real user IDs

The restart function for real user IDs is specified in the RESTART operand of the USER statement.

```
USER userid-name ... RESTART=YES | NO
```

If this is generated as RESTART=YES then the type of client and any generated sign-on services will also play a role in service restart:

- If a sign-on service has been generated for a client that signs on using this user ID, then this service will control whether a service restart is performed or whether the open service is terminated abnormally, see the description of the sign-on service in the openUTM manual “Programming Applications with KDCS”.
- If a terminal or TS application does not sign on via a sign-on service, then openUTM always initiates a service restart.
- If a UPIC client does not sign on via a sign-on service then the UPIC client must explicitly initiate the restart service, otherwise the open service is terminated abnormally see the manual “openUTM-Client for the UPIC Carrier System”.
- When using OpenCPIC clients, restart is only possible with cooperative processing (functional unit not equal to "Commit"). The OpenCPIC client must explicitly initiate the restart service, otherwise the open service is terminated abnormally, see manual “openUTM-Client for the OpenCPIC Carrier System”.

Restart function for connection user IDs

If individually generated TS applications sign on via implicit (created by openUTM) connection user IDs, then the restart function is controlled by the RESTART operand in the LTERM statement:

```
LTERM ltermname ... RESTART=YES | NO
```

This parameter is irrelevant for the service restart if the TS application is signed on via an explicitly generated connection user ID or a real user ID.

5.1.10 USP headers for output messages to socket connections

In order that the UTM application is able to communicate with the TS application via the socket interface, a UTM socket protocol (USP) is used on top of TCP/IP. openUTM uses this protocol to convert a bytestream received via the socket interface into a message. The partner application must issue the protocol and prefix it with the input message as the protocol header. openUTM does not usually create a protocol for output messages.

It is possible to set each generation option so that openUTM also prefixes a protocol header for output messages. This is specified in the PTERM or TPOOL statement using the operand USP-HDR=:

- USP-HDR=ALL ensures that openUTM prefixes all output messages of this connection (dialog, or asynchronous message, K message) with a protocol header.
- With USP-HDR=MSG the protocol header is prefixed for K messages only.
- USP-HDR=NO means that no protocol header is prefixed for output messages.

The structure of the protocol header is described in the openUTM manual “Programming Applications with KDCS”.

5.1.11 Code conversion

During communication between the UTM application and a client it may be that the two communication partners are using different codes (EBCDIC, ASCII/ISO8859-1). To ensure that the communication between the partners remains simple despite this, you can generate an automatic code conversion for the following client. This will convert ASCII/ISO8859-1 to EBCDIC and vice versa:

- B** ● BS2000/OSD: TS applications of type SOCKET
- X/W** ● Unix systems, Windows systems:
 - X/W** – OpenCPIC clients and TS applications of type SOCKET and APPLI.
 - X/W** – Server-server communication with LU6.1 and OSI TP partners

You must remember that only printable messages may be exchanged, as binary data may become errored if converted.

The code conversion is controlled using the operand MAP=:

- B** ● BS2000/OSD:
 - B** PTERM/TP00L ... MAP= USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4
 - B** Under BS2000/OSD, you can use various conversion tables. The operator of a UTM
 - B** application under BS2000/OSD can also create their own tables.
- X/W** ● Unix system / Windows system:
 - X/W** PTERM/TP00L ... MAP = USER | SYSTEM (TS applications)
 - X/W** OSI-CON ... MAP = USER | SYSTEM (OSI TP partner)
 - X/W** SESCHA ... MAP = USER | SYSTEM (LU6.1 partner)
 - X/W** Under Windows systems and Unix systems it is possible to modify the conversion table.

By default code conversion is not used, or in other words MAP=USER. If MAP=SYSTEM or SYS1, ..., SYS4, openUTM converts the data from ASCII to EBCDIC and vice versa. Conversion tables are provided for code conversion.



Additional information about code conversion can be found in the appendix on [page 613](#).

5.1.12 Providing address information

For TS applications, remote UPIC clients and OpenCPIC clients, address information is required to establish a connection. This information is stored in the UTM generation. For socket applications, there is no difference between BS2000/OSD and Unix systems or Windows systems. For other clients, the characteristics of the transport system take effect. This information is therefore divided into separate paragraphs.

Port number 102 for TCP/IP connections

For connections via TCP/IP, applications are addressed using port numbers, where the port number 102 plays a special role. You must bear the following in mind:

- B** – In BS2000/OSD the port number 102 is often used, in particular because a BCMAP entry is not required for port number 102.
- B**
- X/W** – In Unix systems and Windows systems, you may not use port number 102.

5.1.12.1 Providing the address information for clients of type SOCKET

When communicating with TS applications via TCP/IP the socket interface is used directly.

The address information required for communication is, for the most part, provided in the UTM generation. openUTM obtains the remaining information from the hosts file/database. Thus no BCMAP entries on BS2000/OSD are required.

KDCDEF generation

The address information is stored in the operands LISTENER-PORT=, T-PROT= and PRONAM= of the BCAMAPPL and TPOOL/PTERM statements.

- BCAMAPPL statement

You must always specify a BCAMAPPL statement for socket applications. The following applies:

- In LISTENER-PORT= you must always enter a port number under which the UTM application waits for the requests of the socket application. The port number must always correspond to the settings of the communication partner.
- In T-PROT= you must always enter SOCKET.
- LISTENER-ID= assigns the connection an optional listener ID. The values for the LISTENER-ID of non-socket connections and socket connections may be assigned independently of each other.

X/W
X/W
X/W

- PTERM statement

If you generate the socket applications individually, you will need to specify the following operands and parameters:

- If the socket application is to establish the connection, the PTERM name must have the format PRT*nnnnn*, where *nnnnn* is the port number from which the socket application establishes the connection. The name may be supplemented by leading zeros.
- In BCAMAPPL= enter the application name as defined in the BCAMAPPL statement.
- In PRONAM= you must always specify the first TCP/IP host name that is generated in the data resources (hosts file, DNS, ...). If you want to use the "mapped host name" function, then you must specify the UTM host name here (see [page 122](#)). You must not specify any other names, i.e. aliases, found in the database (hosts file, DNS, ...).
- In the LISTENER-PORT= operand, you must specify the port number at which the socket application is waiting for connection establishment requests.

- TPOOL statement

If you link the socket application using a LTERM pool, then:

- In PRONAM= you must always specify the first TCP/IP host name that is generated in the data resources (hosts file, DNS, ...). If you want to use the "mapped host name" function, then you must specify the UTM host name here (see [page 122](#)). You must not specify any other names, i.e. aliases, found in the database (hosts file, DNS, ...).

X/W
X/W

If you enter PRONAM=*ANY, then clients from any computer can sign on assuming they are of the same type as specified in PTYPE= .

- In BCAMAPPL= enter the application name as defined in the BCAMAPPL statement.

Example

The following example uses the port number 10100. The socket application is linked via the LTERM pool and runs on the computer PCSOCK01.

```
BCAMAPPL BSPSOCK -
  ,LISTENER=PORT=10100 -
  ,T-PROT=SOCKET -
  ....
TPOOL ...
  ,PTYPE=SOCKET -
  ,BCAMAPPL=BSPSOCK -
  ,PRONAM=PCSOCK01 -
  ....
```

5.1.12.2 Providing address information for clients of type UPIC and APPLI in BS2000/OSD

B For RFC1006 (or ISO) connections you must always generate a separate application name.
B This can be used for all RFC1006 (or ISO) connections regardless of client type.

B KDCDEF generation

B ● BCAMAPPL statement

B In T-PROT= enter either ISO or RFC1006 (these two values are treated identically in BS2000/OSD). You may **not** specify T-PROT=SOCKET.
B

B ● PTERM statement

B If you want to generate the clients individually, specify the following operands and parameters:
B

B – Under PTERM name, you must enter the name that was defined for this client when the network was generated.
B

B – In BCAMAPPL= enter the application name defined in the BCAMAPPL statement.

B – In PRONAM= enter the name of the computer on which the client is running. This name is specified when the network is generated.
B

B – In PTYPE= you must enter either UPIC-R or APPLI, the option PTYPE=*ANY is not permitted.
B

- TPOOL statement

If you want to link clients via the LTERM pool, then:

- In BCAMAPPL= enter the application name that is defined in the BCAMAPPL statement.
- In PRONAM= you can enter the name of the computer on which the clients are running. If you specify PRONAM=*ANY the clients can sign on from any computer as long as it is of the type specified in PTYPE=.
- In PTYPE= you must enter either UPIC-R or APPLI, the option PTYPE=*ANY is not permitted.

B Coordination with the BCAM generation

If a client is linked to a UTM application via RFC1006 and uses a port number that is \neq 102, then a BCMAP entry is required for this client:

```

B /BCMAP FUNCT=DEFINE ,SUBFUNCT=GLOBAL ,NAME=name-of-client , -
B /      ES=partner-processorname ,PPORT#=listener-port-no , -
B /      PTSEL-I=tselector-of-client

```

5.1.12.3 Providing address information for clients of type UPIC and APPLI in Unix systems and Windows systems

X/W Under Unix systems and Windows it is possible to write your own BCAMAPPL statement
X/W for an application name that is generated using MAX APPLINAME=. This statement can
X/W then be used to specify all the parameters you require.

X/W To link clients via RFC1006, see [section “Providing address information for the CMX
X/W transport system \(Unix systems and Windows systems\)” on page 110](#). The KDCDEF
X/W generation must contain all the necessary address information.

X/W KDCDEF generation for RFC1006**X/W ● BCAMAPPL statement**

X/W – *appliname*: You can select any application name, but the name must be unique within
X/W the network, as KDCDEF uses it to create a T-selector.

X/W – If OPTION CHECK-RFC1006=YES then a port number must be specified for
X/W LISTENER-PORT.

X/W In all other cases, the default value is 0 (no port number).

X/W – In T-PROT you **must always** enter RFC1006.

X/W – In TSEL-FORMAT= enter the format indicator for the name that you have defined
X/W as the *appliname* (see above). It is recommended that you always make an entry for
X/W the operand TSEL-FORMAT=.

X/W ● PTERM statement

X/W If you want to generate the client individually via a PTERM statement, enter the
X/W following:

X/W – As PTERM name use the T-selector of the client. The client on the client computer
X/W must be entered with this T-selector as the local application.

X/W – In BCAMAPPL= enter the application name as defined above.

X/W – In LISTENER-PORT= enter the port number which the client is reached as output
X/W port on the client computer.

X/W – In PRONAM= you must always enter the first TCP/IP host name of the client
X/W computer, as generated in the data resources (hosts file, DNS,...). If you want to use
X/W the "mapped host name" function, then you must specify the UTM host name here
X/W (see [page 122](#)). You must not specify any other names, i.e. aliases, found in the
X/W database (hosts file, DNS, ...).

X/W – In PTYPE= you must specify wither UPIC-R or APPLI.

X/W ● TPOOL statement

X/W If you want to connect the client via an LTERM pool, enter the following:

X/W – In BCAMAPPL= enter the application name as defined above.

X/W – In PRONAM= you can enter the name of the computer on which the client is
 X/W running. This must be the first TCP/IP host name that is listed in the data resources
 X/W (hosts file, DNS,...). If you want to use the "mapped host name" function, then you
 X/W must specify the UTM host name here (see [page 122](#)). You must not specify any
 X/W other names, i.e. aliases, found in the database (hosts file, DNS, ...).

X/W If you enter PRONAM=*ANY the clients can sign on from any computer as long as
 X/W it is of the same type specified in PTYPE=.

X/W – In PTYPE= you must enter either UPIC-R or APPLI.

X/W *Example*

X/W The following example uses the port number 10030 locally and the remote application has
 X/W the port number 12030. The UPIC client runs on a computer called PCUPR.

X/W BCAMAPPL BSPUPR –
 X/W ,LISTENER-PORT=10030 –
 X/W ,T-PROT=RFC1006 –
 X/W ,T-SEL-FORMAT=T –
 X/W

X/W PTERM UPRPART –
 X/W ,PTYPE=UPIC-R –
 X/W ,BCAMAPPL=BSPUPR –
 X/W ,PRONAM=PCUPR –
 X/W ,LISTENER-PORT=12030 –
 X/W ,T-PROT=RFC1006 –
 X/W ,T-SEL-FORMAT=T –
 X/W

X/W The statements for a TS application are created analogue, except that you must specify
 X/W PTYPE=APPLI in PTERM.

5.1.12.4 Additional information for LTERM pools in Unix systems and Windows systems

X/W An LTERM pool can be used by any partner application on a specific computer to establish
X/W a connection to the UTM application, if the partner application is of the appropriate type
X/W (PTYPE). If PRONAM=*ANY is generated in the TPOOL statement, then TS applications
X/W of the generated type can connect to the UTM application from any computer.

X/W The TPOOL statement does not specify a name (station name) for this communication
X/W partner. The UTM application determine this name from the Transport Name Service that
X/W is supplied when the connection is established.

X/W The following procedure is used:

X/W ● If a TS application is communicating with the UTM application, an attempt is made to
X/W find out the computer name using the local Name Service.

X/W ● If the TS application is communicating with the UTM application via the Internet, an
X/W attempt is made to find out the computer name using the Internet Name Service.

X/W ● If no name can be located for the computer the network process assigns it the name
X/W *ANY.

X/W ● Then an attempt is made to obtain the T-selector from the transport address. If a
X/W T-selector is found then it is used as the station name.

X/W ● If no T-selector can be found, then the station name 'NETM $nnnn$ ' is used for the
X/W TS application. $nnnn$ stands for a number between 0000 and 9999 and is automatically
X/W incremented by openUTM.

X/W It often makes sense to communicate with LTERM pools via TCP/IP connections, if LTERM
X/W pool is generated with processor names (TPOOL ...,PRONAM=).

5.1.12.5 Providing address information for OpenCPIC clients in BS2000/OSD

B The address information for OpenCPIC clients is stored in the ACCESS-POINT and
B OSI-CON statements. This information must match that in the OpenCPIC generation.

B KDCDEF generation

B ● ACCESS-POINT statement

B – In *access_point_name*, define the name of the local OSI TP access point.

B – In TRANSPORT-SELECTOR= specify the BCAM name of the local UTM appli-
B cation.

B – If necessary, you must also enter the session and presentation selector of the local
B OSI TP access point in SESSION-SELECTOR= and PRESENTATION-
B SELECTOR=. If these selectors are not required, enter *NONE in both cases.

B ● OSI-CON statement

B – In LOCAL-ACCESS-POINT= enter the name of the OSI TP access point as defined
B above.

B – In NETWORK-SELECTOR= enter the name of the computer on which the
B OpenCPIC client is running.

B – In TRANSPORT-SELECTOR= enter the BCAM application name of the OpenCPIC
B client.

B – If necessary, you must also enter the session and presentation selector of the local
B OSI TP access point in SESSION-SELECTOR= and PRESENTATION-
B SELECTOR=. If these selectors are not required, enter *NONE in both cases.

B Coordination with the BCAM generation

B If an OpenCPIC client is connected to the UTM application via RFC1006 and uses port
B number \neq 102, then this client requires a BCMAP entry:

```
B /BCMAP FUNCT=DEFINE, SUBFUNCT=GLOBAL, NAME=name-of-client, –
B /      ES=partner-processormame, PPORT#=listener-port-no, –
B /      PTSEL-I=tselector-of-client
```


5.1.12.6 Providing address information for OpenCPIC clients in Unix systems and Windows systems

X/W The address information for OpenCPIC clients is stored in the ACCESS-POINT and OSI-CON statements. This information must be coordinated with both the OpenCPIC generation.

X/W To connect clients via RFC1006 is described in [section “Providing address information for the CMX transport system \(Unix systems and Windows systems\)” on page 110](#). The KDCDEF generation must contain all the required address information.

X/W KDCDEF generation for RFC1006

X/W The KDCDEF generation must be coordinated with the client generation, see the example shown on [page 165](#).

X/W ● ACCESS-POINT statement

X/W – In *access_point_name* define the name of the local OSI TP access point.

X/W – In TRANSPORT-SELECTOR= you must specify the T-selector of the local OSI TP access point. This name must be coordinated with the generation on the client side.

X/W – In TSEL-FORMAT= enter the format indicator of the T-selector. It is recommended that you always provide a value for the TSEL-FORMAT= operand.

X/W – If necessary, you must also enter the session and presentation selectors of the local OSI TP access point in SESSION-SELECTOR= and PRESENTATION-SELECTOR=. This name must be coordinated with the generation on the client side.

X/W If these selectors are not required, you must specify *NONE in both cases.

X/W – In LISTENER-PORT= enter the port number under which the OpenCPIC client contacts the local UTM application.

X/W – In T-PROT= you must always specify RFC1006.

- OSI-CON statement
 - In LOCAL-ACCESS-POINT= enter the name of the OSI TO access point as defined above.
 - In NETWORK-SELECTOR= you must enter the name of the computer on which the OpenCPIC client is running. This must always be the first TCP/IP host name that is generated in the data resources (hosts file, DNS, ...). If you want to use the "mapped host name" function, then you must specify the UTM host name here (see [page 122](#)). You must not specify any other names, i.e. aliases, found in the database (hosts file, DNS, ...).
 - In TRANSPORT-SELECTOR= you must specify the T-selector of the client. This name must be coordinated with the generation on the client side.
 - In TSEL-FORMAT= enter the format indicator of the T-selector. It is recommended to always provide values for the TSEL-FORMAT= operand.
 - If necessary, you must also enter the session and presentation selectors of the OpenCPIC client in SESSION-SELECTOR= and PRESENTATION-SELECTOR=. This name must be coordinated with the generation on the client side.
If these selectors are not required, you must specify *NONE in both cases.
 - In LISTENER-PORT enter the port number of the OpenCPIC client.
 - In T-PROT you must always specify RFC1006.

5.1.13 Examples of the generation of a client/server cluster

The following examples show how to connect a UPIC client which is running on a Windows system PC to a UTM application in BS2000 and Unix systems.

B Example 1: Connecting a UPIC client to openUTM under BS2000/OSD

B The UTM server application is located on a host with the name BS2HOST1, the client
B program is running on a PC with the computer name PCCLT002. The transport connection
B is to be established via TCP/IP (address format RFC1006).

B ● UTM generation under BS2000/OSD

B *** Define BCAM application name for the UTM server application:***
B BCAMAPPL SERVER, T-PROT=RFC1006

B *** Generate client:***
B PTERM UPICTTY, PTYPE=UPIC-R, LTERM=UPICLT, BCAMAPPL=SERVER, -
B PRONAM=PCCLT002
B LTERM UPICLT

B *** Define TAC for the client:***
B TAC TAC1, PROGRAM=SERVICE

B The statement LTERM UPICLT means that when signing on openUTM implicitly uses a
B connection user ID called UPICLT.

B When establishing a link via the RFC1006 protocol, no BMAP entries are required.

B ● Entries in the side information file (upicfile) of the openUTM client

B * UTM application under BS2000/OSD
B SDsamplaw SERVER.BS2HOST1 TAC1
B * or, if you require automatic conversion of user data
B * from ASCII to EBCDIC and vice versa
B HDsamplaw SERVER.BS2HOST1 TAC1

B ● Specification in the client program

B Enable_UTM-UPIC "UPICTTY"
B Initialize_Conversation "samplaw"

X Example 2: Connecting an UPIC client to openUTM under Unix system

X This example describes the TCP/IP RFC1006 connection of a UPIC client to a UTM application under Unix system. The example shows the coordination of the generation for both communication partners.

X The UTM application is running on a computer with the name UXHOST01. The client is located on a Windows system for which the name PCCLT001 has been generated in the KDCDEF. The UTM application receives the local port number 1230 and the client has the port number 1240.

X ● Generating the UTM server on the UNX computer

X BCAMAPPL UTMUPICR, LISTENER-PORT=1230, T-PROT=RFC1006, TSEL-FORMAT=T

X PTERM UPICTTY, PTYPE=UPIC-R, LTERM=UPIC, BCAMAPPL=UTMUPICR, PRONAM=PCCLT001, \
X LISTENER-PORT=1240, T-PROT=RFC1006, TSEL-FORMAT=T

X LTERM UPIC, USER=UPICUSER

X USER UPICUSER, PERMIT=ADMIN

X The statement USER UPICUSER... is used to generated an explicit connection user ID.

X ● Entries in the side information file of the client computer

X * Local application

X LNUPICTTY UPICTTY PORT=1240

X * UTM application under Unix system with port 1230, TCP/IP host
X name=UXHOST01

X SDsampladm UTMUPICR.UXHOST01 KDCHELP PORT=1230

X ● Entries in the hosts file of the client computer

X In the HOSTS file of the Windows system you must make the following entry for the mapping of the computer name UXHOST01 to the IP address of the Unix system:

X *ip-address* UXHOST01

X ● Specification in the client program

X Enable_UTM_UPIC "UPICTTY"

X Initialize_Conversation "sampladm"

B/X Example 3: Connecting an OpenCPIC client

B/X An OpenCPIC client is running on a Unix system with the host name UNIXPRO1. The client
 B/X connects itself via RFC1006 to a UTM application under BS2000/OSD and to a UTM appli-
 B/X cation under Unix system. The following is to apply:

- B/X – In the UTM application under BS2000/OSD, the client calls the transaction code
 B/X TRAVEL02 and in the UTM application under Unix system it calls the transaction code
 B/X STATIST1.
- B/X – It is to be possible to have up to 10 parallel connections to BS2000, and up to 2 parallel
 B/X logical connections to Unix system.
- B/X – The UTM application under BS2000/OSD uses the local port number 102. The UTM
 B/X application under Unix system uses the local port number 12000.
- X – The OpenCPIC client uses local port number 13000.

B ● UTM generation under BS2000/OSD

B UTMD APT = (2, 7, 16, 2)

B ACCESS-POINT SERVER,
 B T-PROT = RFC1006,
 B P-SEL = *NONE,
 B S-SEL = *NONE,
 B T-SEL = C'UTMSERV1',
 B AEQ = 1

B OSI-CON CONNECTB,
 B LOCAL-ACCESS-POINT = SERVER,
 B P-SEL = *NONE,
 B S-SEL = *NONE,
 B T-SEL = C'CPICCLT1',
 B N-SEL = C'UNIXPRO1',
 B OSI-LPAP = OSILPAPB

B OSI-LPAP OSILPAPB,
 B APT = (2, 7, 16, 4),
 B APPLICATION-CONTEXT = UDTSEC,
 B AEQ = 1,
 B ASS-NAMES=CPIC,
 B ASSOCIATIONS=10,
 B CONTWIN=0

B TAC TRAVEL02 ...

B When linking via the RFC1006 protocol using port number 102, no BCMAP entries are
 B required.

```

X ● UTM generation under Unix system
X   UTMD APT = (2, 7, 16, 3)
X   ACCESS-POINT STATSERV,
X     T-PROT = RFC1006,
X     P-SEL = *NONE,
X     S-SEL = *NONE,
X     T-SEL = C'UTMSERV2',
X     LISTENER-PORT = 12000,
X     T-PROT = RFC1006,
X     T-SEL-FORMAT = T,
X     AEQ = 1
X   OSI-CON CONNECTX,
X     LOCAL-ACCESS-POINT = STATSERV,
X     P-SEL = *NONE,
X     S-SEL = *NONE,
X     T-SEL = C'CPICCLT1',
X     N-SEL = C'UNIXPRO1',
X     LISTENER-PORT = 13000,
X     T-PROT = RFC1006,
X     T-SEL-FORMAT = T,
X     OSI-LPAP = OSILPAPX
X   OSI-LPAP OSILPAPX,
X     APT = (2, 7, 16, 4),
X     APPLICATION-CONTEXT = UDTSEC,
X     AEQ = 1,
X     ASS-NAMES=CPIC,
X     ASSOCIATIONS=2,
X     CONTWIN=0
X   TAC STATIST1 ...
B/X ● OpenCPIC generation
B/X   *** Entry for the local application
B/X   LOCAPPL OPENCPI,
B/X     APT = (2, 7, 16, 4),
B/X     AEQ = 1
B   *** Connection to UTM application under BS2000/OSD
B   PARTAPPL UTMSBS20,
B     APT = (2, 7, 16, 2),
B     APPLICATION-CONTEXT = utm-secu,
B     AEQ = 1,
B     ASSOCIATIONS = 10,
B     CONTWIN = (10,10),
B     CONNECT = 10

```

```

R    *** TAC in the UTM application under BS2000/OSD
B    SYMDEST TRAVEL,
B    PARTNER-APPL = UTMSBS20,
B    PARTNR-APRO = TRAVEL02

X    *** Connection to UTM application under Unix system
X    PARTAPPL UTMSUNIX,
X    APT = (2, 7, 16, 3),
X    APPLICATION-CONTEXT = utm-secu,
X    AEQ = 1,
X    ASSOCIATIONS = 2,
X    CONTWIN = (2,2),
X    CONNECT = 2

X    *** TAC in the UTM application under Unix system
X    SYMDEST STATIST,
X    PARTNER-APPL = UTMSUNIX,
X    PARTNR-APRO = STATIST1

```

B/X ● TNS entries in the OpenCPIC client computer (tnsxfrm format)

```

B/X  OPENCPI\
B/X   PSEL V''
B/X   SSEL V''
B/X   TSEL RFC1006 T'CPICCLT1'
B/X   TSEL LANINET A'13000'

B    UTMSBS20\
B    PSEL V''
B    SSEL V''
B    TA RFC1006 ip-address-bs2 PORT 102 T'UTMSERV1'

X    UTMSUNIX\
X    PSEL V''
X    SSEL V''
X    TA RFC1006 ip-address-unix PORT 12000 T'UTMSERV2'

```

5.2 Generating printers (on BS2000/OSD and Unix systems)

W
W



Printers cannot be generated in UTM applications under Windows systems.

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

Printers that are to be used by a UTM application are connected via LTERM partners that are configured with the logical properties for printers. LTERM partners for printers are defined in the LTERM statement. Printers **cannot** be connected via LTERM pools. Physical printers are defined with the PTERM statement, which is also where the assignment is made to the LTERM partner.

B/X
B/X
B/X

The connection setup by openUTM can be defined either with PTERM...,CONNECT=YES or with LTERM...,PLEV=. The connection can also be established using administration functions. See also the openUTM manual “Administering Applications”.

B/X
B/X
B/X



LTERM statement on [page 355](#)

The most important properties for LTERM partners via which printers can connect to an application are defined with the following operands:

B/X

- Itermname

B/X

Name of the LTERM via which the printer is connected to the UTM application.

B/X

- CTERM=

B/X

Defines the printer control LTERM so that the user can administer printers, print jobs, and the print jobs in the message queue of the LTERM partner.

B/X

B/X

- PLEV=

B/X

Number of printer messages for which openUTM attempts to establish a connection to the printer assigned to this LTERM partner.

B/X

B/X

– If PLEV=1, a connection is established for each print job.

B/X

– If PLEV=*n*, the connection is established for the *n*th print job (n=1 to 32767).

B/X

B/X

– If PLEV=0, the connection setup is not initiated by pending print jobs, rather is initiated explicitly by the administrator using the command

B/X

KDCLTERM...ACT=CON or KDCPTERM.

B/X

B/X

The connection is shut down again as soon as there are no further messages for this printer.

B/X

B/X PLEV= makes it easier for the user to use printers from various UTM applications
 B/X (printer sharing). In this case, the connection between a UTM application and the
 B/X printer is only kept open while the print job is being transmitted, in order to allow
 B/X other applications to establish a connection. If there are unprocessed print
 B/X messages for a printer in the message queue of the LTERM partner when the UTM
 B/X application terminates, these print messages are retained until the next application
 B/X start. Before terminating the application, the administrator can initiate the
 B/X processing of the outstanding print messages using the command KDCAPPL
 B/X SPOOLOUT=ON.

B/X ● QAMSG=

B/X Messages to printers can be buffered in the message queue of the LTERM partner,
 B/X even if the printer is not connected to the application.

B/X ● USAGE=O

B/X USAGE=O defines a printer as a communication partner which can connect to an
 B/X application via the LTERM partner. Messages can only be sent from the application
 B/X to the printer.

B/X Each printer must be described in the configuration, i.e. for each printer, a PTERM
 B/X statement is written with the physical properties of the printer and the assignment is made
 B/X to an LTERM partner. The assignment between the printer and LTERM partner is static, i.e.
 B/X the assignment applies until it is canceled using administration commands. You can assign
 B/X another printer (of the same type) to an LTERM partner during operation, e.g. in the event
 B/X of a printer fault.

B/X  **PTERM statement** on [page 437](#)

B/X The most important properties for printers are defined with the following operands:

B/X ● ptermname

B/X Name of the printer

B *BS2000/OSD:*

B The BCAM name or the RSO name of the printer must be specified.

X *Unix systems:*

X The name of a printer group of the spool system must be specified. A printer group
 X used by UTM applications should comprise only **one** printer. This is the only way to
 X ensure that all parts of a message are output to the same printer if a message
 X comprises message segments (see also the information on printer pools).
 X

- B/X

 - CID=

The printer is assigned a printer ID *printer_id* via which the printer can be identified by a printer control LTERM. The printer control LTERM attaches to the LTERM partner to which the printer is assigned.
- B/X
B/X
B/X

 - CONNECT=

Specifies whether or not openUTM automatically establishes a connection to the printer when the application starts. The printer is then explicitly occupied by the application until the next time the connection is cleared down, even if there are no print jobs.
- B/X
B/X
B/X
B/X

 - LTERM=

Name of the LTERM partner assigned to the printer *ptermname* and via which the printer is connected to the UTM application.
- B/X

 - PTYPE=

BS2000/OSD
Printer type or *RSO.

Unix systems:
Printer type.

To output the data, the printer process (*utmprint*) calls the *utmlp* script. The call also passes parameters to *utmlp* in addition to the data to be printed. By default, *utmlp* then passes the data to the lp command (see PTYPE=PRINTER on [page 453](#)).
- B
B

 - USAGE=O (only BS2000/OSD)

The communication partner is a printer.

B/X The maximum values and limits that are to apply throughout the application for printers are
B/X defined with the MAX statement.

B/X  **MAX statement** on [page 368](#)

B/X The default and maximum values relevant throughout the application for printers are
B/X defined with the following operands:

B/X ● CONRTIME=

B/X Time in minutes after which openUTM makes cyclical attempts to reestablish a
B/X logical connection. openUTM attempts this for:

B/X – Printers to which openUTM establishes a connection as soon as the number of
B/X print jobs for this printer exceeds the generated threshold value
B/X (LTERM...,PLEV>0). When the connection is aborted, the number of print jobs
B/X must be greater than or equal to the threshold value if openUTM is to attempt
B/X to re-establish the connection.

B/X – Printers to which openUTM automatically establishes a connection
B/X (PTERM...,CONNECT=YES), provided that the connection was not terminated
B/X by the administration.

B/X If no connection is established when the application starts or if the connection is
B/X interrupted during operation, openUTM attempts to reestablish the connection at
B/X intervals of CONRTIME=.

B/X ● PGPOOL=

B/X A sufficiently large value must be specified for the PGPOOL operand so that the
B/X page pool can accommodate all print messages and does not overflow in the event
B/X of a high print volume.

B/X ● TRMSGLTH=

B/X Maximum length of physical messages that can be exchanged between the UTM
B/X application and printers.

B ● LOGACKWAIT=

B Maximum time in seconds that openUTM is to wait for an acknowledgment from the
B output devices. This acknowledgment is


- B – for a printer, the logical print acknowledgment from the printer,
- B – for an RSO printer, the acknowledgment from RSO,
- B – with an FPUT call to another application, a transport acknowledgment.

5.2.1 Generating RSO printers (BS2000/OSD)

B Via the OLTP interface of RSO (remote spool output), openUTM obtains access to all
 B printers that support RSO, i.e. including printers connected via LAN or PC. openUTM does
 B not establish a transport connection to these printers, rather serves them via the OLTP
 B interface, i.e. openUTM reserves the printer for RSO and transfers the print job to RSO.

5.2.1.1 Entries for the KDCDEF generation

B To print to an RSO printer from openUTM, the desired printer is defined in the generation
 B under its RSO name as an RSO printer in the PTERM statement. The printer must be
 B defined and activated on the RSO side. This section only lists the RSO specific statements
 B and operands, the other printer-specific parameters are described on [page 168f](#).

B  **PTERM statement** on [page 437](#)
 B RSO printers served by openUTM via the OLTP interface are defined with the
 B following operands:

B ● ptermname

B For an RSO printer, the name of the printer must be specified here as it was defined
 B in RSO (logical RSO device name).

B ● PTYPE=

B PTYPE=*RSO is specified as the printer type. No particular printer type is specified
 B for RSO printers. openUTM obtains the printer type in accordance with the RSO
 B device information in the RSO call.

B ● PRONAM=

B For an RSO printer, *RSO must be specified as the processor name.

B  **MAX statement** on [page 368](#)
 B Limit values relevant in the application for printers are defined with the following
 B operands:

B ● TRMSGLTH=

B Maximum length of the physical messages that can be exchanged between the
 B UTM application and printers.
 B If RSO printers are defined, REMOTE-BUFFER-SIZE in the SPOOL parameter file
 B must be greater than TRMSGLTH.

5.2.1.2 Entries for RSO and SPOOL

B In order for openUTM to use the OLTP interface of RSO, RSO version 3.0A and higher and
 B the software products required by RSO must be installed. The RSO subsystem must be
 B active. If you have specific questions, read the RSO manual:

B Device definition


B With the UTM tool SPSEIVE, you open the SPOOL parameter file for the printer definition.
 B The system administrator must configure the printer in RSO for UTM print jobs:

```
B ADD-SPOOL-DEVICE...ADMINISTRATOR=*ADMINISTRATOR(...),
B           PROCESSING-CONTROL=*PAR(
B                               DISCONNECTION=*YES
B                               RESET={*YES | *NO }
B                               CONTROLLER-START=AT-PRINTER-START)
```

- B ● If a new printer is configured, up to 8 RSO device managers can be entered with the
 B parameter ADMINISTRATOR=*ADMINISTRATOR(...). An RSO device manager can
 B modify a device with MODIFY-SPOOL-DEVICE or start a DEVICE with START-
 B PRINTER-OUTPUT.
- B ● It is advisable to work with the parameter DISCONNECT=*YES, because with
 B SOCKETS the printer shuts down the connection when the time set on the printer has
 B expired.
- B ● The parameter CONTROLLER-START must be set to AT-PRINTER-START.
- B ● If RESET=*YES, the settings of the printer menu are used. This also applies regardless
 B of the device entry if openUTM is working with formats. If “logical” formats are used (see
 B note at the end of the section) then openUTM behaves as if no formats are used.
 - B – If a format name is transferred by openUTM with an FPUT in the KCMF field, a
 B RESET=*YES is sent by default to the printer by FHS before the message, so that
 B the menu setting of the printer comes into effect before printing. In the printer menu
 B you can set various fonts or CPI values, for example. In this case, RSO processes
 B a message with format names as per the setting CONTROL=TRANSPARENT.
 - B – If no format name is transferred by openUTM with an FPUT in the KCMF field, RSO
 B only sends a RESET to the printer before the message if RESET=*YES is entered
 B in the device definition. If no RESET was sent to the printer, the applicable values
 B are the printer menu values currently set on the printer, which may have been
 B changed by a previous print job. RSO handles a printer message without format
 B names as per the setting CONTROL=PHYSICAL.

B The commands ADD-SPOOL-FORM for form entries and ADD-SPOOL-CHARACTER for character sets have no effect on UTM print jobs. If both UTM RSO print jobs and RSO SPOOLOUT are processed under the same logical RSO device name, forms and character sets are only relevant for the latter. As the only printer control character, RSO adds the RESET character string for UTM print jobs.

B A UTM print job to an RSO printer is not placed in the SPOOL queue.

B  As of FHS V08.2C formats can be output to **all** RSO printers as long as the formats have been created appropriately (as of IFG V8.1B), see the manual “FHS V8.2C - formatting system for openUTM, TIAM, DCAM”. Formats can be created in such a way that they are sent to RSO as non-device-specific formats.

B Sample device entry

B Output of a device entry under which the printer is defined for RSO:

```

B /show-spool-dev PGTP0041,inf=*all
B DEVICE-NAME          : PGTP0041
B DEVICE-TYPE         : 9021RP
B ACCESS-DATE        : 2006-11-27
B ----- DEVICE-ACCESS -----
B DEVICE-ACCESS      : *TCP-ACCESS
B ACCESS-TYPE        : *TACLAN
B PROCESSOR-NAME     : *NONE
B STATION-NAME       : *NONE
B MNEMONIC-NAME      : *NONE
B PROGRAM-NAME       : *NONE
B INTERNET-ADDRESS   : PGTP0041
B PORT-NAME          : 9100
B LPD-PRINTER-NAME   : *NONE
B FROM-PORT-NUMBER   : 0
B TO-PORT-NUMBER     : 0
B ----- TWIN-DEVICE-DEF -----
B SLAVE-MNEMONIC-NAME : *NONE
B ESD-SIZE            : 0
B ----- DEVICE-INFORMATION -----
B FORMS-OVERLAY-BUFFER: 32767
B CHARACTER-SET-NUMBER: 64
B ROTATION            : NO
B DUPLEX-PROCESSING  : NO
B FORMS-OVERLAY      : NO
B RASTER-PATTERN-MEM : *NONE
B TRANSMISSION       : IGN
B FONT-TYPE          : IGN
B FACE-PROCESSING    : NO
B MAXIMUM-INPUT-TRAY : 1
B MONJV              : NO
    
```

```

R   NOTIFICATION           : NO
B   ENCRYPTION             : NO
B   UNICODE                : NO
B   SUPP-FORMAT-NAME      :
B   TEXT
B   PLAIN-TEXT
B   ----- ADMINISTRATOR -----
B   USER-IDENTIFICATION   : *NONE
B
B   IDENTIFICATION        : OEC MW 135
B   TERMINAL              : PROCESSOR-NAME      :
B                           STATION-NAME       :
B   ----- SPOOLOUT-CONTROL -----
B   SHIFT                 : 0
B   LINE-FEED-COMPRESS    : YES
B   BLANK-COMPRESS        : YES
B   START-FORM-FEED       : YES
B   FORM-FEED             : *SINGLE-SHEET
B                           DEFAULT-TRAY-NUMBER : 1
B                           OUTPUT-TRAY-NUMBER  : 0
B   SKIP-TO-CHANNEL       : OPTIM
B   SKIP-TO-NEXT-PAGE     : BY-FORM-FEED
B   ESCAPE-VALUE          : NONE
B   ----- PROCESSING-CONTROL -----
B   CONTROLLER-RESERVED   : NO
B   FORM-NAME             : STD
B
B   DISCONNECTION         : YES
B   BUFFER-SIZE           : 1024
B   RESET                 : YES
B   REPEAT-MESSAGE        : TYPE                : SYS
B                           LIMIT                : NO
B                           RETRY-TIME          : GLB
B   RESTART-ACTION        : LIMIT                : NO
B                           RETRY-TIME          : GLB
B   SYNCHRONIZATION       : PRINTER
B   TIMEOUT-MAX           : 2
B   PAGE-EJECT-TIMEOUT    : NO
B   BAND-IDENTIFICATION   : *NONE
B   LOAD                  : NO
B   MODULO2               : NO
B   RECOVERY-RULES        : *SYSTEM
B   POLLING               : NO
B   PRINTER-PARAM-FILE    : *SYSTEM
B   RESOURCE-FILE-PREFIX  : *SYSTEM
B   CONTROLLER-START      : AT-PRINTER-START
B   ----- CHARACTER-SET-POS -----
B   POSITION-1             : N-U

```

```

R   POSITION-2           : N-U
B   POSITION-3           : N-U
B   POSITION-4           : N-U
B   POSITION-5           : N-U
B   POSITION-6           : N-U
B   POSITION-7           : N-U
B   POSITION-8           : N-U
B   POSITION-9           : N-U
B   POSITION-10          : N-U
B   POSITION-11          : N-U
B   POSITION-12          : N-U
B   POSITION-13          : N-U
B   POSITION-14          : N-U
B   POSITION-15          : N-U
B   POSITION-16          : N-U
B   ----- MISCELLANEOUS -----
B   REDIRECTION-DEVICE  : *NONE
B   LANGUAGE-EXT-TYPE   : *SYSTEM
B   LINE-SIZE           : 150
B   CHARACTER-IMAGE     : *NONE

```

B **Defining the RSO buffer size**

B To be able to print out messages of any length, the RSO buffer must be greater than or equal to the maximum message length in openUTM (MAX ...,TRMSGLTH=). Since the maximum value for the UTM buffer size is 32 KB, the RSO buffer size in a session in which openUTM is running, must be adapted to this value:

```
B /MODIFY-SPOOL-PARAMETER...SPOOLOUT-OPTIONS=*PAR(REMOTE-BUFFER-SIZE=32)
```

B A smaller value can also be selected if smaller values are generated for openUTM. Any modification will come into effect in the next SPOOL session.

B **VTSU codes**

B When UTM messages containing VTSU codes are output to printers connected directly via BCAM, openUTM calls the VTSU program in order to convert VTSU codes into printer-dependent escape sequences. If UTM messages are output to RSO printers, the conversion of the VTSU codes is carried out by RSO. An extensive adaptation to the known VTSU control characters was striven for in this case. Additional information can be found in the RSO manual.

5.2.1.3 Activating printers for openUTM

B Each printer used in an RSO session must be started with START-PRINTER-OUTPUT. A
 B START-PRINTER-OUTPUT statement can be contained in an ENTER file which is
 B processed automatically when the RSO subsystem starts, or the system administrator or
 B RSO device manager uses this statement to explicitly start the printer after the subsystem
 B has started.

B If you want to print to an RSO printer from openUTM, the printer must be released for
 B openUTM:

B /START-PRINTER-OUTPUT DEVICE=NAME=*RSO(NAME=devicename,
 B ALLOWED-ACCESSES='UTM')
 B or ALLOWED-ACCESSES=('RSO','UTM'))

5.2.1.4 Querying printer information

B *Printer information in openUTM*

B The administration command KDCINF can be used in openUTM to query the current printer
 B status. If required, the connection to the printer can be set up/shut down or locked using the
 B administration commands KDCPTerm and KDCLTerm or via the program interface for
 B administration. See also the openUTM manual "Administering Applications".

B *Printer information in RSO*

B Using BS2000 information functions, users and RSO device managers can output the
 B printer status of the RSO printers with the following command:

B /SHOW-SYSTEM-STATUS INFORMATION=*REMOTE(DEVICE=NAME)

5.2.1.5 Releasing printers in the event of an error

B If the automatic repetition of the print job by openUTM and RSO was not successful in the event of an error, the RSO device manager can temporarily release the printer with the command

```
B /STOP-PRINTER-OUTPUT DEVICE-NAME=RSO-PRINTER(NAME=rso-printer,STOP=IMMEDIATE)
```

B so that the printer can then be reserved again for openUTM with the START-PRINTER-OUTPUT command.

B The parameter TRACE=YES can be specified under \$TSOS for diagnostic purposes. In this case, a trace is generated and is stored under \$SYSSPOOL.SYSTRC.RSO.*devicename.date.time*:

```
B /START-PRINTER-OUTPUT DEVICE-NAME=*RSO(NAME=devicename,
B                               TRACE=YES,
B                               ALLOWED-ACCESSES=('RSO','UTM'))
```

B See also the "RSO" manuals.

5.2.2 Generating printer pools

B/X A printer pool is made up of several printers (=printer groups in Unix systems) assigned to **one** LTERM partner. A PTERM statement with the same *ltermname* for PTERM...,LTERM= is written for each printer in the pool.

B/X openUTM distributes the print output as evenly as possible to the printers in the pool.

B/X Messages that are made up of message segments are always output in full by openUTM to **one** printer or printer group (in Unix systems) in the pool.

5.2.3 Bypass mode (BS2000/OSD)

B With a locally connected printer, the term bypass mode is also used instead of spool mode. Bypass mode is possible if the terminal can conduct dialog independently of the print output. Bypass mode must only be implemented for terminal types 975x and 9763 or a corresponding emulation (see the manual "MT9750, 9750 Emulation under Windows")

5.2.4 Generating printer control LTERMs

- B/X In the generation you can define printer control LTERMs so that users themselves can
B/X administer the default printers and print job queues even without administration authori-
B/X zation, e.g. delete the current print job.
- B/X Each printer is assigned an LTERM partner, which is configured for an output medium
B/X (LTERM...,USAGE=O). All output jobs for this printer are “sent” by openUTM to the
B/X message queue of the associated LTERM partner, which thus becomes the print job queue.
B/X It is also possible to assign several printers to an LTERM partner (printer pool). In this case,
B/X all printers work with this print job queue.
- B/X A print control LTERM is an LTERM partner which is configured as a dialog partner
B/X (LTERM...,USAGE=D). Via this LTERM partner, a client or a terminal user can connect to
B/X the application in order to administer printers and the associated print job queues.
- B/X You assign the printers to the respective printer control LTERM via the LTERM partner, i.e.
B/X for the LTERM partner you specify the printer control LTERM to which the printers are
B/X assigned with LTERM...,CTERM=*printercontrol-ltermname*.
- B/X To enable the printer control LTERMs to identify the printers assigned to them, you assign
B/X a control identification (CID) to each printer in the PTERM statement. This CID must be
B/X unique within the area of a printer control LTERM, because the printer control LTERM
B/X addresses the printers using the printer ID. It is particularly important for the printer IDs to
B/X be unique in the case of printer groups. Each printer in the pool must be assigned a
B/X separate printer ID which does not belong to any other printer in the printer control LTERM.
- B/X To restrict access to the printer control LTERM to a particular group of users, you can assign
B/X a lock code to the printer control LTERM just like any other LTERM partner.
- B/X An acknowledgment procedure is used for the printers assigned to a printer control LTERM.
B/X This procedure can be switched on and off as required for each individual printer. All
B/X printers assigned to a printer control always run in automatic mode with their first appli-
B/X cation start after a regeneration.
- B/X For further information on printer administration, see the openUTM manual “Administering
B/X Applications”.

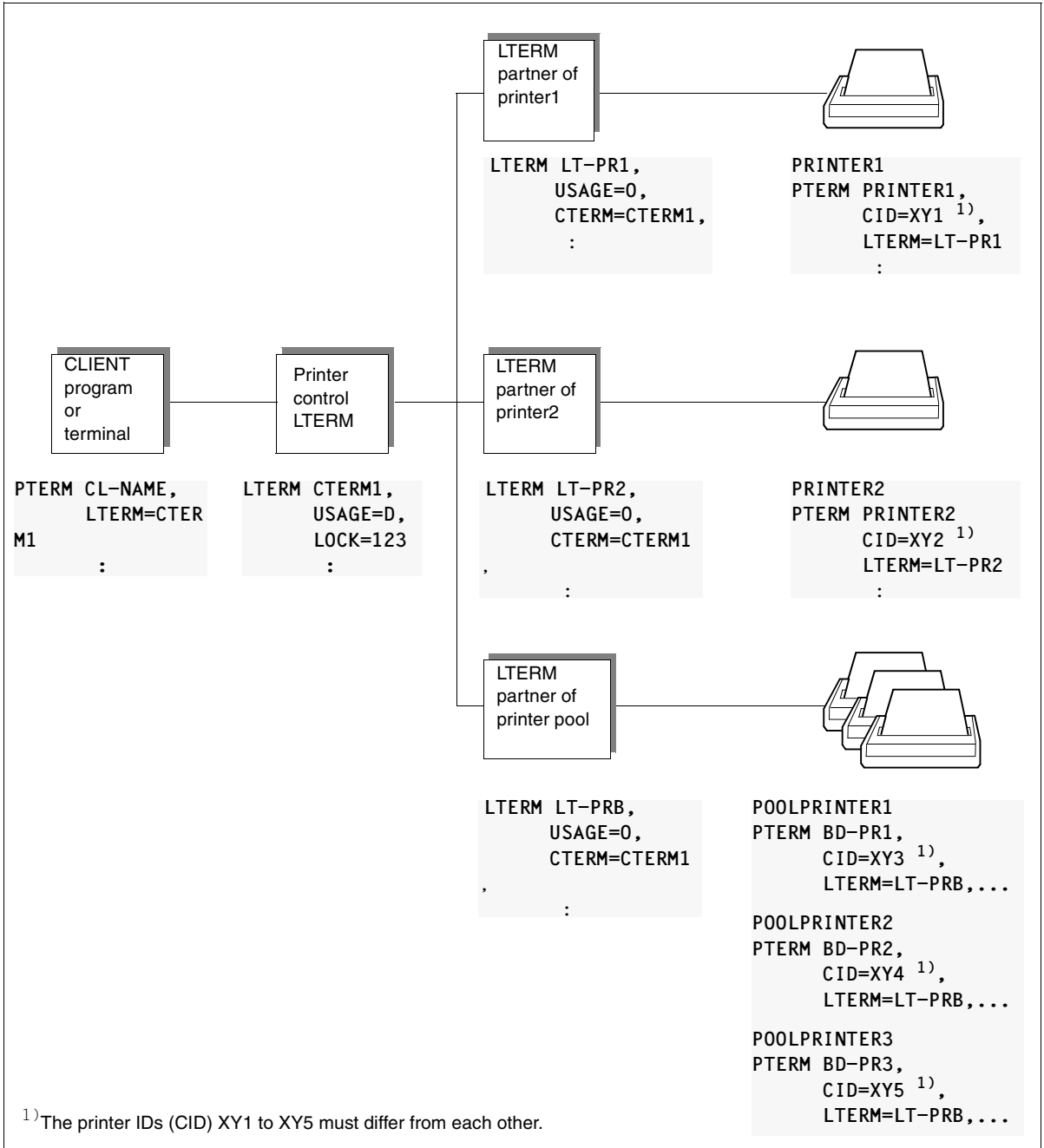


Figure 14: Configuring a printer control LTERM and the associated printers

5.3 Generating service-controlled queues

openUTM offers service-controlled queues, that is, message queues the processing of which is controlled by the program units of the application. A program unit of a dialog or asynchronous service must read the message of a service-controlled queue itself using the KDCS call DGET. A service may also be designed to wait for the arrival of a message.

Since the messages are saved in the page pool, you must ensure that the page pool is configured at a sufficient size.

openUTM provides three different service-controlled queue types:

- USER queues (user-specific)
- TAC queues (defined using TAC statements)
- temporary queues (created using QCRE calls and deleted using QREL calls)



A general introduction to service-controlled queues and their application scenarios can be found in the openUTM manual “Concepts und Functions”.

The implementation of the application scenarios is described in the openUTM manual “Programming Applications with KDCS”. This also contains information about processing service-controlled queues (reading, writing and deleting).

5.3.1 USER queues

Each user of a UTM application is automatically provided with a permanent message queue which is addressed using the user ID.

For USER queues it is possible to generate the data access control which prevents reading or writing by using Q-READ-ACL or Q-WRITE-ACL (USER statement).



USER statement on [page 530](#)

The following operands are available for USER queues:

- QLEV=

QLEV can be used to prevent the page pool becoming overloaded with messages for this USER.

QLEV specifies the maximum number of asynchronous messages that may be buffered in the USER queue (default setting: 32767, i.e. no limit). If the specified value is exceeded, the subsequent behavior is determined by the value in the QMODE parameter.

- **QMODE=**
Determines the behavior of openUTM in the event that the USER queue has already exceeded the maximum number of permitted messages that may be buffered and has thus reached the Queue level (operand QLEV=). If the value STD is set all new DPUT calls are rejected, if WRAP-AROUND is set the oldest message is overwritten by the new message.
- **Q-READ-ACL=**
Specifies the read and delete authorizations in the USER queue for external users. If you do not specify Q-READ-ACL= all users have read and delete authorization in the queue.
- **Q-WRITE-ACL=**
Specifies the write authorization in the USER queue for external users. If you do not specify Q-WRITE-ACL= all users have write authorization in the queue.

5.3.2 TAC queues

The generation of transaction codes with TYPE=Q (queue) creates permanent Message Queues with fixed names.

The TAC queue with the fixed name KDCDLETQ is called the dead letter queue. openUTM provides this TAC queue to save queued messages sent to transaction codes or TAC queues that could not be processed (see [page 490](#)).

TAC queues may be locked for reading or writing (see [page 224](#)).



TAC statement on [page 483](#)

The following operands are important for the generation of a queue defined using TAC statements:

- *tacname*
Name of the TAC.
- **TYPE=Q**
TYPE=Q must be specified for TAC queues. A Message Queue is generated. It is possible to use an FPUT or DPUT call to write a message to a queue of this nature, or to use a DGET call to read a message from the queue.

- ADMIN=
Specifies whether access to this queue requires administration authorization.
- DEAD-LETTER-Q=
Specifies whether queued messages in this message queue are to be saved in the dead letter queue if not processed correctly when the maximum number of attempts to redeliver the message (MAX statement, REDELIVERY parameter) has been reached.
- QLEV=
QLEV can be used to ensure that the page pool is not overloaded by jobs for this TAC queue.
QLEV specifies the maximum number of asynchronous messages that may be in the Message Queue of this transaction code.
- QMODE=
Determines the behavior of openUTM in the event that the maximum permitted number of messages is already saved in a queue and thus the queue level is reached.
- Q-READ-ACL=
The key set defines the authorizations that permit reading or deletion of messages in this queue.
- Q-WRITE-ACL=
The key set defines the authorizations that permit writing messages to this queue.
- STATUS=
Specifies whether the message queue is locked or released when the application is started.

5.3.3 Temporary queues

Temporary queues are created and deleted dynamically by program calls. The name of the queue may be determined by the user or assigned by openUTM.

The maximum number of temporary queues is defined within the QUEUE statement.



QUEUE statement on [page 458](#)

The following operands are used to define temporary queues:

- **NUMBER=**
Specifies the maximum number of temporary queues that may exist at any one time during an application run ($1 \leq \text{NUMBER} \leq 500\,000$).
- **QLEV=**
QLEV can be used to prevent the page pool becoming overloaded with messages for this temporary queue.
QLEV specifies the default value for the maximum number of messages that may exist in a temporary messages queue at any one time (default value: 32767, in other words an unlimited queue length).
- **QMODE=**
Determines the behavior of openUTM in the event that the maximum permitted number of messages is already saved in a temporary queue and thus the queue level is reached.

5.3.4 Specifying the maximum waiting time for reading from service-controlled queues

At generation it is possible to set the maximum length of time that a service is permitted to wait for a message for a queue. This maximum wait time may be defined separately for the dialog and asynchronous services (MAX statement, QTIME operand). This ensures that a terminal user or client does not have to wait several minutes for the system to react to an error in a UTM program unit, or that a resource remains blocked for too long.



MAX statement on [page 368](#)

The following operands are used to specify the maximum length of time that a service is permitted to wait for a message in a service-controlled queue:

- QTIME=

Specification of the maximum waiting time in seconds (default: 32767 seconds). You can define separate maximum values for the waiting time for the dialog or asynchronous services.

If a greater waiting time is specified in a program unit run than specified in QTIME then openUTM resets the waiting time to the value defined here.

5.3.5 Limiting the maximum number of redeliveries to service-controlled queues

At generation you can define whether a message to a service-controlled queue is to be placed back in the queue if the transaction in which the message was read is reset. You can also limit the number of redeliveries at generation (MAX statement, REDELIVERY operand). This prevents, for example, endless loops if a program error occurs.



MAX statement on [page 368](#)

The maximum number of redeliveries of messages to service-controlled queues is defined using the following operand.

- REDELIVERY= (... ,number2)

number2 is the maximum number of redeliveries of messages to a service-controlled queue ($0 \leq \textit{number2} \leq 255$).

Values between 0 and 254 indicate the number of redeliveries. The value 255 means that the message can be redelivered any number of times.

Default 255, i.e. the number of redeliveries is unlimited.

5.4 UTM messages

openUTM generates UTM messages that inform about certain events or request dialog input. The UTM messages are located in a message module which is supplied with openUTM (standard message module).

You can modify the messages of openUTM using the message tools KDCMTXT and KDCMMOD, and create own message modules (user message modules) which are adapted to your own needs.

You can adapt the standard UTM messages by:

- modifying UTM message texts (e.g. translation into other languages)
- deleting or adding UTM message destinations
- modifying inserts

You must declare user message modules in the configuration using the MESSAGE statement.



The entire event reporting mechanism and the tools KDCMTXT and KDCMMOD are described in detail in the openUTM manual “Messages, Debugging and Diagnostics”.

5.4.1 Messages in openUTM under BS2000/OSD

B The following components of UTM event reporting are included in the delivery package:


- B** ● the German standard message module KCSMSGSGS
- B** ● the English standard message module KCSMSGSE
- B** ● the message definition file SYSMSH.UTM.061.MSGFILE

B The message definition file contains the message texts in German and English and forms the basis for the creation of user message modules.

B You must declare user message modules in the configuration using the MESSAGE statement. If you do not issue a MESSAGE statement, the German standard message module KCSMSGSGS is used to output messages

B In order to internationalize your application you can create multiple user specific message modules in a variety of languages and include them in the configuration of a UTM application. In this way, UTM messages can be output to a terminal user in a variety of languages within any given UTM application. The language used to communicate with the user depends on the locale (language identifier *lang_id* and territorial identifier *terr_id*) that you assign the user during generation as well as on the availability of the user message module which has been assigned an appropriate locale during generation.

B If more than one message module is assigned for a UTM application, then a locale must be
B assigned to each message module.

B  **MESSAGE statement** on [page 406](#)
B User message modules are defined with the following operands:

B ● MODULE=


B Name of the message module you want to incorporate in the configuration.

B ● LIB=

B Identifies the object module library from which the message module is loaded
B dynamically. If a generated message module *modulename* is not contained under the
B name *lmodname* in the library *omlname* when linking the application, the linkage
B editor reports that the module is missing. The message module can be loaded
B dynamically.

B ● LOCALE=

B Defines the language environment (locale) of the message module if language-
B specific message modules have been created for specific message output. These
B national-language message modules are used for users and LTERM partners
B whose language and territorial identifiers match the locale defined here. For further
B information, see [section “Internationalizing the application – XHCS support \(BS2000/OSD\)” on page 237](#).
B

B  **USER statement** on [page 530](#) and **LTERM statement** on [page 355](#)
B With the following operand you specify the message module (and the language)
B which is used to output messages to the user/client:

B ● LOCALE=


B Language environment (locale) of the user/client.

B *Application message module and user message modules*

B If multiple message modules are used for an application, then a distinction is drawn
 B between application and user message modules. The application message module is the
 B message module in whose MESSAGE statement the locale specifications correspond to
 B those in the MAX statement. The application message module has a special significance
 B within the application. The message destination specifications entered for the application
 B message module determine the destination for message output. The message destination
 B specifications in the other message modules have no significance. The application
 B message module is used to output messages to the message destinations SYSLST,
 B SYSOUL and CONSOLE.

B Messages to the destinations STATION, SYSLINE and PARTNER employ the message
 B module whose *lang_id* and *terr_id* specifications (of the Locale) correspond to those of the
 B user or LTERM partner for which the message is output. Here, the user specification takes
 B priority over the LTERM partner specification, i.e. if a user is signed on when the message
 B is output, openUTM uses the message module which corresponds to this user.

B If a locale (*lang_id*, *terr_id*) for which there is no message module in the application has
 B been generated for a user or LTERM partner, then the user or LTERM partner is assigned
 B a message module which corresponds with the *lang_id* and for which no *terr_id* has been
 B generated. If no such message module is present, the application message module is used
 B to output messages to this user or LTERM partner.

B  **MAX statement** on [page 368](#)
 B With the following operand you specify the message module which is used as appli-
 B cation message module:

- B ● LOCALE=

B The locale of the message module that is to be used as the application message
 B module. A message module with this locale must be generated with a MESSAGE
 B statement.

5.4.2 Messages in openUTM under Unix systems and Windows systems

X/W The following components of UTM event reporting are included in the delivery package:

- X/W ● Standard message module of openUTM

X/W The standard message module contains the text for the standard messages in English
 X/W and standard settings for the message destinations (e.g. terminals, SYSLOG file).
 X/W openUTM only generates the messages from the standard message module if no NLS
 X/W message catalogs and no user message modules exist for a language.

X/W
 X/W
 X/W



CAUTION!

The standard message module must be linked in **each** UTM application program.

X *Unix systems*

X The standard message modules `kcsmsgs.o` (K and P messages) and `kcxmsgs.o`
 X (U messages) are supplied with openUTM on Unix systems. Both are contained in the
 X library `utmpath/sys/libwork.a` or `utmpath/sys/libwork.so`.

X The expression “standard message module” is used for both modules.

W *Windows systems*

W The standard message module is supplied in the `utmpath\sys\kcsmsgs.obj` object file.


- X/W ● Message definition file `msgdescription` (in the `utmpath`)

X/W It contains the standard message texts in German and English, as well as the
 X/W framework definitions for the UTM messages (structures of messages).

- X/W ● NLS standard message catalogs (Unix systems) / message DLLs (Windows systems)

X/W NLS standard message catalogs are supplied with openUTM in German and in English.
 X/W Under Windows systems the message catalogs are implemented as message DLLs.

X/W The message catalogs only contain the message texts. When structuring the messages
 X/W from an NLS catalog, openUTM uses the structure information and message destina-
 X/W tions of the default message module, or if available, the user message module.

- X *Unix systems*
- X Under Unix systems, the NLS standard message catalogs are stored in the directories
X *utmpath/nls/msg/xxx*. In this case, *xxx* is the language ID for the corresponding
X language.
- X On Unix systems you can modify existing NLS message catalogs and create your own
X NLS message catalogs for other languages.
- X You can set the language to be used for the messages to your preferred language in the
X LANG shell variable.
- W *Windows system*
- W Under Windows system, the message DLLs are stored in the directories
W *utmpath\nls\msg\xxx*.
- W On Windows systems you can change the message destinations with your own
W message module, but you cannot change the text of the messages.
- W You can set the language to be used for the messages to your preferred language in the
W LANG shell variable.
- X/W In the simplest case, you operate your application with the standard UTM messages, i.e.
X/W you do not modify the UTM messages nor the UTM message destinations. In this case, no
X/W additional specifications are required in the KDCDEF generation. You must merely link the
X/W standard message module *utm-directory/sys/kcmsmsg.o* to the application program.
- X/W When generating an application, you use the MESSAGE statement to define the name of
X/W the message module. This message module is then created using a C source file written by
X/W KDCMMOD.
- X/W  **MESSAGE statement** on [page 406](#)
X/W Use the following operand to define the message module when generating the
X/W application:
X/W
 - MODULE=
- X/W Name of the module that is to be created using the tool KDCMMOD.
- X/W To modify the standard messages, use the message tools KDCMTXT and KDCMMOD (see
X/W openUTM manual “Messages, Debugging and Diagnostics under Unix systems and
X/W Windows systems”).

5.4.3 User-specific message destinations

In addition to the message destinations, CONSOLE, SYSOUT etc., there are also four so-called user-specific message destinations. The user can define up to four message destinations of their own. These message destinations are named using USER-DEST-*number* and may be user queues, TAC queues, asynchronous TACs or LTERM partners.



This makes it, among other things, possible to display the K and P messages of your application to the administrator at the WinAdmin administration workstation (see also „WinAdmin Online-Hilfe“).

Messages indicating warning level violations cannot however always be delivered to their user-specific message destination.

The new KDCDEF statement, MSG-DEST, is used to agree the user-specific message destinations.



MSG-DEST statement on [page 412](#)

Using the following operands you can agree a maximum of four user-specific message destinations:

- *msg-destination*
Refers to the message destination with the specification USER-DEST-*number* (*number*=1..4). Message destinations must be assigned to the messages using the KDCMMOD.
- NAME=
Specifies the name of a user or TAC queue or an asynchronous TAC or LTERM partner to which the messages are to be sent (this name must be defined using a TAC, USER or LTERM statement).
- DEST-TYPE=
Type of message destination (USER queue, TAC or LTERM).
- MSG-FORMAT=
Specifies the format of the messages that are to be sent. Only the inserts of a non-printable format (FILE; default) or the inserts and messages texts of a printable format (PRINT) are transferred.

Assigning the message destination USER-DEST-*number*

Messages that openUTM is to output to a message destination, USER-DEST-*number*, must also be assigned to this message destination using the utility KDCMMOD (MODMSG statement).

5.5 Message distribution and multiplexing with OMNIS (BS2000/OSD)

B The services of the BS2000/OSD software product OMNIS can be used for UTM applications in BS2000/OSD. OMNIS is a Session Manager that enables a terminal user to call the services of various UTM applications directly, even if the UTM applications are distributed in the network. In this case, the terminal user need not know the processor nor the UTM application in which the service is located. OMNIS automatically establishes a connection to the “correct” UTM application and controls the assignment of messages (message distribution).

B When implementing OMNIS, you can also use the multiplex function provided by openUTM in BS2000/OSD: a large number of terminals can be connected to a UTM application via a small number of transport connections.

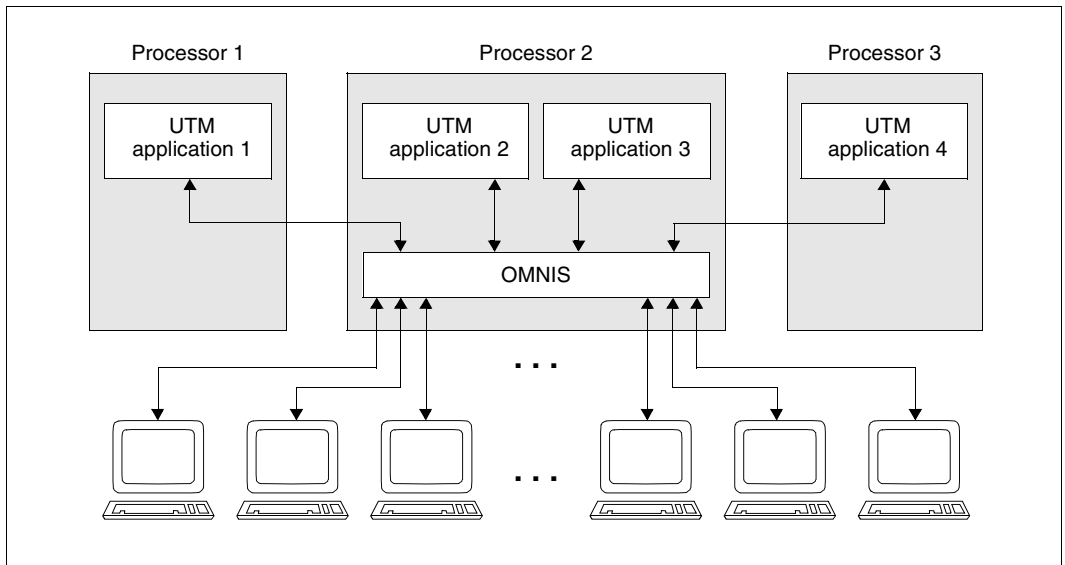


Figure 15: Message distribution and multiplexing with OMNIS

B See also the manuals “OMNIS/OMNIS-MENU Functions and Commands” and
B “OMNIS/OMNIS-MENU Administration and Programming”.

B **i** OMNIS Version 6.3A or later supports XHCS (Extended Host Code Support, see
B [page 237](#)), OMNIS V7.0A or later also for UTM partners with multiplex protocol. The
B prerequisite for using XHCS is that OMNIS is running on a processor with
B DCAM ≥ V10.0 and VTSU-B ≥ V10.1B.

5.5.1 Multiplex connections

- B In normal dialog mode, one transport connection exists between a terminal and a UTM application on the processor. In order for a user to be able to call the services of an application, the user must open a session with the application, i.e. a communication relationship between two addressable units in the network. A session setup generally means that the user must provide identification to the application. This can also occur implicitly.
- B OMNIS now offers you the option of connecting simultaneously to several UTM applications, even on different processors. However, you are only actually connected to one communication partner (namely OMNIS). The Session Manager now transmits the input messages (user jobs) to the applications with which you are connected.
- B Transport connections and sessions exist on both links of the communication relationship, i.e. the link from user → Session Manager and from Session Manager → application. This is illustrated in the diagram below:

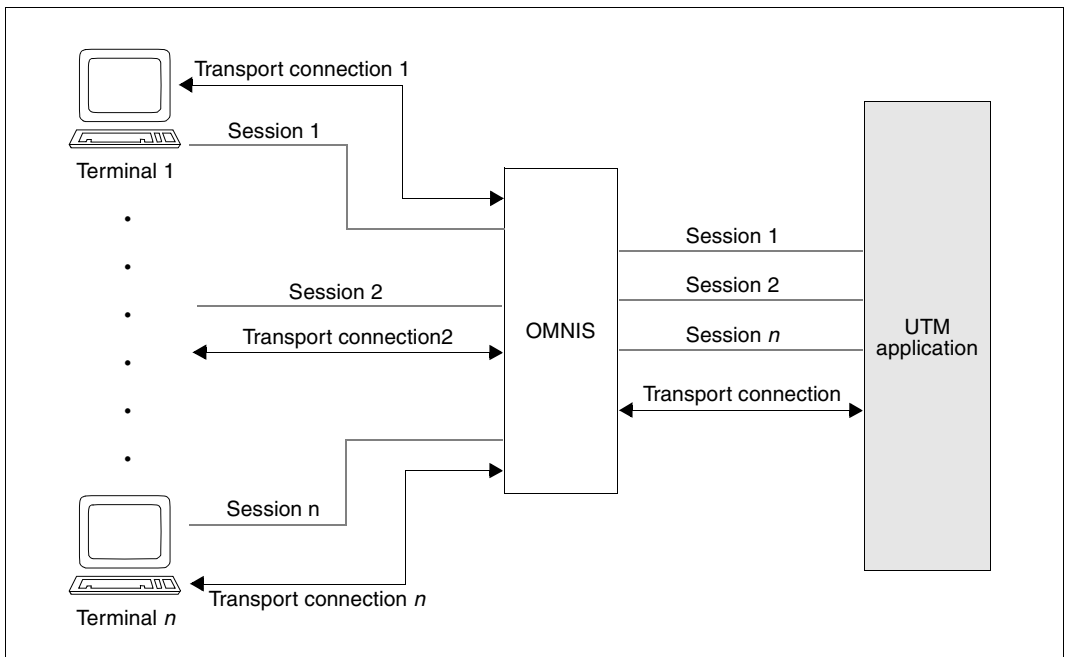


Figure 16: Transport connections and sessions when multiplexing

B A **transport connection** is a connection between two programs or between one program
B and a terminal, via which messages can be exchanged. A transport connection has a
B defined beginning (connection setup) and a defined end (connection shutdown) and is
B known to the transport system.

B A **session** is one of several completely different data streams, which is maintained via a
B transport connection. A session has a defined start (session setup) and a defined end
B (session shutdown) and is known to the transport system. In the special case of OMNIS and
B openUTM, a session is understood to be a communication relationship between a UTM
B application and an OMNIS terminal, which begins with the logical opening of the session
B and ends when the session is closed.

B A one-to-one assignment between transport connection and session exists on the link from
B terminal → Session Manager.

B This one-to-one assignment is cancelled on the link from Session Manager → application
B and several sessions can be assigned to a transport connection. In this way, a number of
B terminals can “multiplex”, i.e. connect to an application via a transport connection. In
B extreme cases, all sessions between the Session Manager and the application can be
B processed via a single transport connection.

5.5.1.1 Defining multiplex connections

B Each multiplex connection must be described with a MUX statement. It is not possible to
B enter multiplex connections dynamically.

B When multiplexing, the communication between the Session Manager and the application
B takes place using the PUTMMUX protocol. The task of this protocol is to enable several
B sessions to be processed via one transport connection and to provide the Session Manager
B with status information on the UTM application under BS2000/OSD.

B A PUTMMUX connection can exist between a UTM application under BS2000/OSD and
B OMNIS as the Session Manager. PUTMMUX connections, also called “multiplex connec-
B tions”, are defined by the MUX statement when generating the UTM application.

B
B
B



MUX statement on [page 414](#)

The most important properties for multiplex connections are defined with the following operands:

B

- *name*

B

Name of the multiplex connection.

B

- BCAMAPPL=

B
B

Local application name of the UTM application, used by the Session Manager to establish the connection to the UTM application.

B

- CONNECT=

B
B

Establishment of a transport connection to the Session Manager when the application starts.

B

- MAXSES=

B
B

Maximum number of simultaneously active sessions between the Session Manager and the UTM application.

B
B
B
B

When establishing a multiplex connection, openUTM and OMNIS negotiate which MUX protocol versions are supported by both sides of the connection. If there are no MUX protocol versions supported by both partners, the multiplex connection is not established (UTM messages K140 and K141).

B

The following restrictions apply with the current definition of the protocol:

B
B
B

- connections between two UTM applications are not supported
- printers are not supported
- only the Session Manager can open a session to a UTM application.

B
B
B
B
B

The add-on product OMNIS-MENU is available if you are using OMNIS in menu-driven mode. OMNIS-MENU enables you to communicate with various UTM applications via a user-friendly, menu-driven interface. For further details, see the manuals “OMNIS/OMNIS-MENU Functions and Commands” and “OMNIS/OMNIS-MENU Administration and Programming”.

5.5.1.2 Outputting asynchronous messages without prior announcement

- B With OMNIS V7.0 or later, openUTM can also send asynchronous messages without prior
 - B announcement to terminals that are connected to the application via a multiplex connection.
 - B In other words, openUTM does not announce that an asynchronous message is pending
 - B for the user by generating UTM message K012, rather it transfers the message immediately
 - B to OMNIS. In this case, OMNIS is responsible for outputting the asynchronous message.
-
- B This means that terminals connected to a UTM application via a multiplex connection can
 - B also open sessions via LTERM partners generated by an LTERM statement or a TPOOL
 - B statement with "ANNOAMSG=N".

5.5.1.3 Confirming the connection shutdown by the partner

- B If a user is connected to a UTM application via a multiplex connection, each of the two
 - B partners – the UTM application or the user – can request the closedown of this session. As
 - B a result of this request, the session switches to the state "DISCONNECT PENDING". The
 - B session is not yet released. The session is not definitively closed until the partner on the
 - B other side confirms the session closedown.
-
- B For a specific length of time (approx. 10 minutes) after the request for session closedown
 - B has been issued, the session can be released by the closedown confirmation of the partner.
 - B Only after this time span has expired can the administrator of the UTM application also
 - B release the session (administration command KDCPTERM).
-
- B From the output of the administration commands KDCINF PTERM and KDCPTERM, the
 - B administrator of the UTM application can determine whether the session is in the state
 - B "DISCONNECT PENDING". See also the openUTM manual "Administering Applications".

5.5.2 Statistics on multiplex connections

B The administrator of the UTM application can use the command

B `KDCINF MUX,OPTION=MONITORING`

B to instruct openUTM to output statistics on multiplex connections. See also the openUTM manual “Administering Applications”. The UTM administrator receives information on:

B ● The utilization level of the multiplex connection.

B Information is supplied on the number of input and output messages exchanged via multiplex connections since the start of the application.

B ● BCAM bottlenecks.

B openUTM supplies information on the number of application messages that could not be accepted by BCAM since the application start due to BCAM bottlenecks, and hence the number of messages openUTM must request be sent again.

5.5.3 Combination of multiplex connections and direct connections

B If you are connecting terminals to your UTM application via direct connections as well as via multiplex connections of the Session Manager, the messages are distributed as follows:

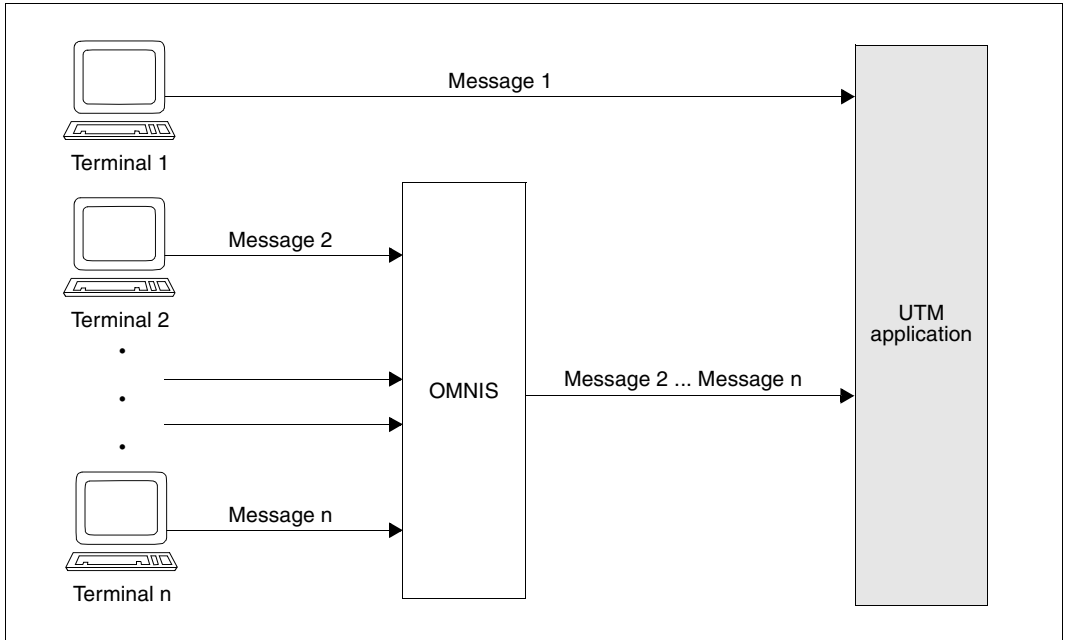


Figure 17: Combination of multiplex and direct connections

B This means that messages via direct connections can overtake messages via multiplex
 B connections. In particular load situations, this leads to shorter response times on the direct
 B connection if a data jam occurs on the multiplex connections. There can be several reasons
 B for this:

B ● The volume of messages from the terminals is so high that the multiplex connections
 B are overloaded.

B ● All UTM processes are occupied with jobs and therefore cannot retrieve all incoming
 B messages immediately.

B There are two ways in which the UTM administrator can avoid the probability of a data jam:


B ● Increase the number of multiplex connections and distribute the volume of messages
 B evenly over these lines.

B ● Increase the current number of UTM processes.

B To guarantee the administrator the fastest possible access to the UTM application at all
 B times, the administrator's terminal should be connected to the application via a direct
 B connection.

5.6 Generating load modules, common memory pools and shared code (BS2000/OSD)

B This section describes how to generate program units, areas and load modules.

B  In the openUTM manual “Using openUTM Applications under BS2000/OSD” you will find more information and recommendations

- B – on structuring an application program
- B – on providing shared code in the system memory or in common memory pools
- B – on the sequence in which modules are loaded and how the external references are resolved
- B – on program exchange during live operation

5.6.1 Generating load modules

B It is only necessary to statically link part of the application to the application program (start B LLM, see the openUTM manual “Using openUTM Applications under BS2000/OSD”). The B other parts of the application program must then be available in the form of dynamically B loadable load modules.

B As early as the KDCDEF generation you must specify at what point in time you want to load B the application parts that are not to statically linked, and to which part of the memory they B are to be loaded. You also specify which program units are to be exchangeable during live B operation.

B The individual load modules of the application must be generated with LOAD-MODULE B statements for BLS implementation. You also specify when the module is to be loaded and B to where. The sequence with which you generate the load modules determines the B sequence in which the load modules are loaded (see LOAD-MODULE statement on B [page 339](#) and in the openUTM manual “Using openUTM Applications under BS2000/OSD”, B loading modules).

B The assignment of objects (program units and shareable data areas) to load modules is B likewise defined in the generation. In the PROGRAM and AREA statements in which B program units or shareable data areas are generated, the name you assigned to the B associated load module in the LOAD-MODULE statement must be specified in the LOAD- B MODULE operand.

**CAUTION!**

openUTM cannot verify whether the assignment defined with the LOAD-MODULE statement and the LOAD-MODULE operand in the PROGRAM and AREA statements corresponds to the actual division of the load modules in the libraries. When dynamically loading the load modules, openUTM relies on the specifications made in the generation. You must therefore ensure that the link procedures you use for the individual parts of the application program correspond with the specifications made in the generation. Otherwise, openUTM cannot guarantee that a required program will be loaded in the working memory with a particular load module.

The load modules are described at generation in the following manner:

**LOAD-MODULE statement** on [page 339](#)

The properties for load modules are defined with the following operands:

- *lmodname*

Name of the load module. This name is used to assign objects to load modules during generation (program units, areas).

For load modules, you must only specify the names of OMs or LLMs. For performance reasons, openUTM does not support dynamic loading using CSECT or ENTRY names.

- LOAD-MODE=

Specifies when a load module is to be loaded, and specifies the memory area to which it is to be loaded. The load modules can be loaded in the standard context to the local task memory, to a common memory pool or to the system memory.

The parts of the application program can be:

- Linked statically to the application program (LOAD-MODE=STATIC)

The part of the application program that is loaded to the standard context of the application using the command `START-EXECUTABLE-PROGRAM` or `LOAD-EXECUTABLE-PROGRAM`.

- Dynamically loaded to the standard context of the local task memory when the application is started (LOAD-MODE=STARTUP).

These should be program units that are continuously required by the UTM application, or which contain external references to shareable parts of the application.

- B
B – Loaded to the standard context of the local task memory at the first call (LOAD-MODE=ONCALL)
- B
B These should be program units that are not continuously required by the application.
- B
B – Loaded to a common memory pool (LOAD-MODE=(POOL,*poolname*,...))
- B
B The common memory pool must be generated with a MPOOL statement (see [page 205](#)).
- B
B The program units that should be loaded to the common memory pool are those that are required by all processes of a UTM application, and which are shareable, for example, the shareable parts of your program unit or also formats or data areas.
- B
B If an LLM contains public and private slices, the public slice is loaded in a common memory pool and the private slice is loaded in the standard context in the local task memory. You can specify whether the non-shareable part is to be loaded when the application is started (LOAD-MODE=(POOL, pool name, STARTUP)) or only then when that program unit is called (LOAD-MODE=(POOL, pool name, ONCALL)). For more information about the generation of shared code see also [page 204ff](#).
- B
B – Loaded to the system memory as a non-privileged subsystem.
- B
B These application parts must be loaded to the system memory by the BS2000 system administrator before the application is started.
- B
B The private slice of a shareable part contained in nonprivileged subsystems can be linked to the static part of an application program, either when the application is started or the first time it is called.
- B
B How to generate the non-shareable parts is described on [page 204](#).
- B ● LIB=
- B Specifies the library from which the load module is to be loaded
- B
B You can specify object module libraries (OML) or program libraries (PL) which contain type R or L elements.
- B ● VERSION=
- B Specifies which version of a load module is to be loaded
- B
B A program library can contain several versions of an element at the same time. You use the version number to define which version of an element is to be loaded.

- **ALTERNATE-LIBRARIES=**
 - B** Specifies whether autolink is to be used for linking
 - B** The shareable parts of the load module are always loaded without using the Autolink function. You can control whether or not the Autolink function is to be used for loading with the **LOAD-MODULE** statement.
 - B** openUTM suppresses the BLS autolink function when loading dynamically and when exchanging programs, if you specify **ALTERNATE-LIBRARIES=NO**. The load module then must only have open external references to program components that already exist in the working memory when this module is loaded.
 - B** For load modules that are generated using **POOL** or **STARTUP**, the sequence of the **LOAD-MODULE** statements at generation is critical for the resolving of open external references at loading. The sequence with which you generate the load modules determines the sequence in which the load modules are loaded.
 - B** **ALTERNATE-LIBRARIES=YES** ensures that runtime system modules that are also required are dynamically linked when an exchange is made. The autolink function may only be used for modules of the runtime system but must *not* be used for user-specific modules because modules loaded with autolink are not unloaded in a subsequent exchange.
- B** Modules that are neither program units of the application program nor data areas (**AREA**) (e.g. the modules of the runtime systems of the programming languages) need not be declared as dynamically loaded modules with the **KDCDEF** generation tool, even if these modules are not linked statically. You can statically link these modules to larger load modules (**LLM**) and need only generate the name of the load module in the **LOAD-MODULE** statement.

B B The assignment of objects (PROGRAM, AREA statement) to load modules (LOAD-MODULE statement) is also defined in the generation.

B B  **AREA statement** on [page 288](#), **PROGRAM statement** on [page 434](#)

B B The assignment to load modules is defined the following operand:

B ● LOAD-MODULE=

B Name of the load module (*lmodname* in the LOAD-MODULE statement), to which the program is linked.


B Program units, modules, and data areas must be linked statically to the application program if the load module to which they are assigned was generated with LOAD-MODE=STATIC or if they are not assigned to any load module.

B The administration modules (e.g the KDCADM administration program) are to be statically linked to the start LLM or to one of their own load modules. This load module must be loaded when the application is started (LOAD-MODE=STARTUP). The same applies to the START, SHUT, INPUT and FORMAT event exits and the BADTAC, MSGTAC and SIGNON event services.

B If specifications for objects in the statements AREA, LOAD-MODULE, MPOOL, PROGRAM and TAC are modified in the generation, only one new KDCFILE need be created. The next application start must then be based on the new KDCFILE.

5.6.2 Generating shared code and common memory pools

B Many compilers offer the option of creating a shareable part when compiling programs. This
 B shareable part need not necessarily be saved in a separate object module, rather can be
 B contained with the non-shareable part in an LLM, which is subdivided into a public and a
 B private slice.

B  If parts of a program unit are to be shareable, this must be taken into account in the
 B programming. For further information, see the openUTM manual “Programming
 B Applications with KDCS” or the appropriate language supplement.

5.6.2.1 Shared code in system memory

B Using the interfaces provided in BS2000/OSD, **shareable parts** of the application program
 B units and parts of the runtime systems can be loaded either as shareable programs in
 B nonprivileged subsystems.

B The shareable modules must be loaded in the memory by the administrator before the
 B application is started. They can be exchanged while the application is running.

B **Non-shareable** parts of the program units must be created as follows:

B ● The entry point of the program unit (it is in the non-shareable part or in the private slice)
 B must be described in a PROGRAM statement and assigned to a load module there
 B using the LOAD-MODULE operand in the PROGRAM statement.

B ● The load module must be generated with a LOAD-MODULE statement with LOAD-
 B MODE={STARTUP | ONCALL}. The load module or its private slice is loaded dynami-
 B cally into the local task memory (class 6 memory) at the start of the application program.
 B The links in the shared code are established dynamically using the external references
 B to the shareable modules.

B The load modules (OM format) containing the shareable modules of the program unit and
 B the load modules containing the non-shareable program components must not occur
 B together in a program library.

B *Example*

```
B PROGRAM NONSHARE,LOAD-MODULE=NAME1,COMP=ILCS
B LOAD-MODULE NAME1,LIB=UTM.PLIB,LOAD-MODE=STARTUP,VERSION=001
```

B NONSHARE is located in the non-shareable part (for LLMs in the private slice) of the
 B program unit.

5.6.2.2 Shared code in common memory pools

B Objects that are not linked statically when linking the application program can be loaded into
 B a common memory pool. In a common memory pool you can dynamically load several load
 B modules.

B A common memory pool must be generated with the KDCDEF statement MPOOL.



MPOOL statement on [page 410](#)

The most important properties for common memory pools are defined with the following operands:

- *poolname*

Name of the common memory pool. This name is used at generation to assign to a pool those load modules whose Public Slice is to be loaded to the pool (see LOADE-MODULE statement).

- SCOPE=

Specifies the scope of the pool (local application with SCOPE=GROUP or global application with SCOPE=GLOBAL).

For each BS2000 user ID, BLS supports a maximum of eight common memory pools with SCOPE=GROUP and eight common memory pools with SCOPE=GLOBAL.

- PAGE=

Hexadecimal address in the form X'xxxxxxxx'.

If global common memory pools with the same contents/names are used in several UTM applications, the parameter PAGE=X'xxxxxxxx' must be specified with the same address in all applications. The address entered using PAGE= is to be selected in such a way that the address area reserved is available in all these applications.

- SIZE=

Specifies the size of the common memory pool.

The size is specified in units of 64 KB. With 24-bit addressing, the size of a common memory pool is always a multiple of 64 KB. With 31-bit addressing, the size of the common memory pool is calculated by $n * 1MB \geq SIZE * 64 KB$ (where n is selected as a minimum).



Only **one** common memory pool should be defined with SCOPE=GROUP. A number of statically linked load modules can be loaded into this pool. This reduces the time required to set up and load the common memory pools and thereby minimizes the time needed to start the application.

B Generating shareable objects that are to be loaded in a common memory pool

B The following section describes how you generate shareable objects that are to be loaded
B in a common memory pool if you are working with BLS.

- B ● For performance reasons, all shareable parts of an application program that are to be
B loaded in a common memory pool should, as far as possible, be combined into **one** load
B module.
- B ● The program's shareable code module created by the compiler must be contained in an
B LLM or OM. LLMs with slices can be generated with a single LOAD-MODULE
B statement:

```
B LOAD-MODULE llm-name ,VERSION=version-  
B ,LOAD-MODE=(POOL ,poolname , {STARTUP|ONCALL})-  
B ,LIB=program-lib-  
B .ALTERNATE-LIBRARIES={YES|NO}
```

B With this statement, the public slice of the LLM is loaded in the common memory pool
B *poolname*, and the private slice is loaded dynamically either when the application starts
B (STARTUP) or when the program is called (ONCALL). Additional PROGRAM state-
B ments are required for the programs of these LLMs that are called by openUTM.

B If a compiler created two separate object modules for the shareable and non-shareable
B part, then should link these modules beforehand to an LLM with slices using the linker.
B You can then generate this LLM as described above.

B Alternately, you can also generate the shareable and non-shareable module using two
B LOAD-MODULE statements. You should avoid this, if possible, because you cannot
B exchange these two modules without having inconsistencies arise.

- B ● A shareable data area which is to be loaded in the common memory pool must be
B described with an AREA statement. The area must then be contained in the load
B module which is generated as follows:

```
B LOAD-MODULE ar-share ,VERSION=version -  
B ,LOAD-MODE=(POOL ,poolname ,NO-PRIVATE-SLICE) -  
B ,LIB=libname
```

B Areas that were assigned the PUBLIC attribute during compilation or by the linker can
B also be linked together beforehand with other modules in one LLM with slices. This LLM
B can be generated in the following manner:

```
B LOAD-MODULE llm-with-slices ,VERSION=version -  
B ,LOAD-MODE=(POOL ,poolname ,STARTUP)-  
B ,LIB=libname
```

B *Example*

B The example assumes that the COBOL85 compiler was used for compiling and that the compiler has saved the objects in an LLM.

B The shareable modules of the COBOL program units PU1 and PU2, and the data module DATAMOD are to be loaded in the local application pool LCPOOL. LCPOOL is to be loaded at address X'020000', occupy 128 KB, and be write-protected.

```

B /MPOOL          LCPOOL,SIZE=2,SCOPE=GROUP,ACCESS=READ,PAGE=X'20000'
B /LOAD-MODULE  LLM-LCPOOL,VERSION=1,
B /              LOAD-MODE=(POOL,LCPOOL,STARTUP),
B /              LIB=libname
B /PROGRAM      PU1      ,LOAD-MODULE=LLM-LCPOOL,COMP=ILCS
B /PROGRAM      PU2      ,LOAD-MODULE=LLM-LCPOOL,COMP=ILCS
B /AREA         DATAMOD,LOAD-MODULE=LLM-LCPOOL

```

B The object modules must be statically linked to the LLM-LCPOOL LLM before the application is started, i.e. you must specify the option BY-ATTRIBUTES(PUBLIC=YES) in the BINDER statement START-LLM-CREATION, whereby the LLM is divided into a public slice and a private slice. The LLM created in this way must be made available in the library *libname*.

5.7 Job control - priorities and process limitations

openUTM provides two methods with which you can control the distribution of released UTM processes amongst the jobs ready for processing. This means that you can affect the order in which openUTM starts the processing of jobs on transaction codes.

By using one of the methods for job control, you can:

- give important jobs higher processing priority
- prevent many jobs of the same type from running at the same time, thereby causing the processing of other jobs to be delayed
- prevent the blocking of job processing due to long-running jobs. Long-running jobs are services whose processing takes an extremely long time, e.g. because their program units are searching through data or they contain program waits (blocking calls such as PGWT).
- in UTM cluster applications, prevent too many tasks from simultaneously accessing memory areas that are available globally in the cluster.

With both methods you must assign TAC classes to the transaction codes that are subject to a specific job control. You can then **alternatively** select one of the two methods for job control between TAC classes:

- **Priority control**
The distribution the processes amongst the TAC classes is controlled by priorities. These priorities are used by openUTM to determine when the outstanding jobs are to be processed. You turn priority control on with the KDCDEF statement TAC-PRIORITIES.
- **Process limitations**
You limit the number of the processes that are allowed to process jobs of a certain TAC class simultaneously, or you specify how many processes are to remain free for processing jobs of other TAC classes. The process number can be specified individually for every TAC class. The KDCDEF control statement TACCLASS is provided for specifying the number of processes.

You must not use the two methods together in an application, i.e. you must not use the control statements TAC-PRIORITIES and TACCLASS together in the KDCDEF generation.

Dividing the transaction codes into TAC classes

openUTM differentiates between a total of 16 TAC classes. There are 8 classes each available for dialog and asynchronous transaction codes, classes 1 through 8 for dialog transaction codes and classes 9 through 16 for asynchronous transaction codes.

You specify the assignment of the transaction codes to the TAC classes in the KDCDEF generation.



TAC statement on [page 483](#)

Operand TACCLASS=

openUTM makes the following assignments for transaction codes to which you have not explicitly assigned a TAC class (no entry in TACCLASS=):

- dialog transaction codes are not assigned to a TAC class
- asynchronous transaction codes are assigned to TAC class 16

You should combine the transaction codes of similar types of services into one TAC class. A TAC class then represents a type of job in your application.

Which jobs are subject to job control?

Generally only jobs that have been placed in a job queue by openUTM are subject to job control.

Jobs for asynchronous transaction codes are always placed in a job queue first before openUTM selects them for processing.

Jobs for dialog transaction codes, on the other hand, are only placed in a queue in bottleneck situations, e.g. when the number of the available processes has been exhausted. If the load on the application is low, then the dialog jobs are processed immediately because they will not block each other significantly and buffering in the queue would appear to slow the system.

For this reason, the methods for job control for asynchronous jobs are always used, while the methods for job control for dialog jobs are only used in bottleneck solutions.

In addition, the following jobs are not subject to job control:

- Jobs for dialog transaction codes that are not assigned a TAC class. These jobs are always started immediately after they have been received from the transport system.
- Jobs for the transaction codes KDCSGNTC, KDCMSGTC and KDCBADTC with which the event services (sign-on service, MSGTAC and BADTACS program) are started.

Controlling the distribution of resources amongst dialog, asynchronous and PGWT processing

In an initial stage of job processing you should - regardless of the methods used for job control - specify the maximum number of processes of the application that are allowed to process asynchronous jobs at the same time or to wait in Program Wait at the same time. In this manner you can prevent the dialog operation of your application from slowing down due to the processing of such jobs.



MAX statement on [page 368](#)

The process numbers are with the following operands generated:

- `ASYNTASKS=(atask_number,...)`

With *atask_number* you specify the maximum number of the processes of the application that may simultaneously process jobs for asynchronous TAC classes.

- `TASKS-IN-PGWT=`

The maximum number of processes of the UTM application in which program units with blocking calls are allowed to run simultaneously. You must specify `TASKS-IN-PGWT > 0` if you want to assign the `PGWT=YES` property to transaction codes or TAC classes.

The values specified for `ASYNTASKS=(atask_number,...)` and `TASKS-IN-PGWT` in the `MAX` statement are maximum values. When starting the application and in application mode, you can lower the number of processes via the administration to adapt to the current situation.

Default setting

If you do not create any TAC classes, i.e. you do not specify the TACCLASS operand in the TAC statement, then openUTM does not perform any special job control.

Program unit runs with blocking calls are not allowed then. Dialog jobs are processed in the order in which they arrive in openUTM.

If you do not issue a TACCLASS or a TAC-PRIORITIES statement in the generation, then openUTM automatically applies the methods used to limit the number of processes. All TAC classes are administrable, i.e. the UTM administrator can specify numbers of processes for the TAC classes.

5.7.1 Job processing via priority control

To activate job control via priorities you must issue the TAC-PRIORITIES statement in the KDCDEF generation. In it you also specify the algorithms with which the individual dialog or asynchronous TAC classes are to be prioritized.



TAC-PRIORITIES statement on [page 505](#)

You specify the algorithms for the priority control with the following operands:

- **DIAL-PRIO=**
Priority with which the available processes of the application are to be distributed amongst the dialog TAC classes.
- **ASYN-PRIO=**
Priority with which processes for the asynchronous TAC classes with ready asynchronous jobs or interrupted asynchronous jobs are to be distributed.

You can select between the **absolute**, a **relative** or the **same** priority for both dialog and asynchronous TAC classes.

The following is always true, regardless of which algorithm you select:

- The TAC class 1 of the dialog TAC classes has a higher or the same priority as TAC class 2, and this has a higher or the same priority as TAC class 3, etc.
- For asynchronous TAC classes, class 9 has a higher or the same priority as TAC class 10, and this has a higher or the same priority as TAC class 11, etc.

If **absolute priority** is selected, then free processes of the application are always assigned the TAC class with the highest priority, meaning 1 (dialog) or 9 (asynchronously) as long as there are jobs waiting for this TAC class. Only after there are no more jobs waiting in the TAC class with the highest priority are waiting jobs of the TAC class with the next lower priority processed. When the load is high, absolute priorities leads to waiting jobs of a TAC class with a lower priority not being processed for a long time. If you want to prevent this, then you should use relative priorities.

If **relative priority** is selected, then jobs from TAC classes with higher priorities are processed more often than jobs from TAC classes with lower priorities, i.e. free processes are more often assigned higher priority TAC classes (e.g. 1) than lower priority TAC classes if there are jobs ready and available for this. If there are jobs available for all TAC classes, then class 1 is serviced twice as often as class 2, and class 2 is serviced twice as often as class 3 (and so on). The same is true for asynchronous TAC classes.

If **same priorities** is selected, then the same number of jobs (if there are any) from every TAC class are processed.

Jobs within the TAC classes, however, whose processing leads to program waits (TACs with PGWT=YES) are only processed if the maximum number of processes allowed to process the PGWT jobs has not yet been reached.

Reserving processes for dialog jobs outside of the TAC classes

When using priority control for the TAC classes, you can limit the number of processes that process the jobs of the TAC classes to keep some processes free for administrative tasks or internal UTM jobs.

This limitation is the same, however, for all asynchronous TAC classes and for all dialog TAC classes.

You limit the maximum number of processes for asynchronous TAC classes with `MAX ASYNTASKS=(atask_number,...)` as described in [“Controlling the distribution of resources amongst dialog, asynchronous and PGWT processing” on page 210](#).

You limit the number of processes for the dialog TAC classes with the `FREE-DIAL-TASKS=` operand of the `TAC-PRIORITIES` statement.

The number of processes specified in `FREE-DIAL-TASKS` is reserved for the processing of jobs that do not belong to any dialog TAC class. These jobs are asynchronous jobs and dialog jobs that are not assigned a dialog TAC class, and in particular are internal UTM tasks (establishing connections, sending acknowledgments, starting the `MSGTAC` routine, etc.). One of the internal UTM tasks is to pick up the incoming jobs for the UTM application at the job market and, if necessary, enter these in the job queues of the application. These “reserved processes” then help to offload the job market. In particular, if many jobs sent to the application come from the network, then this will prevent a backlog in the network that may reach all the way back to the communication partner.

The number of processes you should reserve for this task depends on your application. It is recommended to reserve one or two processes for this task.

You can change the number of free processes via the administration.



See the openUTM manual “Administering Applications”; KDCADMI operation code `KC_MODIFY_OBJECT` with object type `KC_TASKS_PAR`

Example

The following maximum number of processes is specified in the KDCDEF generation:

```
MAX TASKS=7,ASYNTASKS=2  
TAC-PRIORITIES . . . ,FREE-DIAL-TASKS=3
```

If the application is then started with six processes (start parameter `TASKS=6`), then the following process numbers are available:

- Three processes for processing jobs for the dialog TAC classes 1 through 8 (determined by: $TASKS - FREE-DIAL-TASKS = 6 - 3 = 3$)
- Two ($=ASYNTASKS$) processes for processing jobs for the asynchronous TAC classes 9 through 16
- One process for internal UTM tasks and dialog jobs for transaction codes that are not assigned any TAC class (determined by: $FREE-DIAL-TASKS - ASYNTASKS = 3 - 2 = 1$)



For information on the use of TAC priorities in UTM cluster applications, see also the applicable openUTM manual “Using openUTM Applications”, section “Using global memory areas” in the chapter “UTM cluster applications”.

5.7.2 Job processing via process limitation for TAC classes control

Job control via process limitation is generated using the TACCLASS statement. Process limitation depends on the TAC classes, i.e. you can issue a separate TACCLASS statement for every TAC class.



TACCLASS statement on [page 500](#)

You can alternatively specify one of the two following operands to set up process limitation:

- **TASKS=**

The maximum number of processes that are allowed to process jobs for this TAC class.

- **TASKS-FREE=**

The minimum number of processes that are to be kept free for the processing of jobs from other TAC classes or of jobs that are not assigned a TAC class.

In this method the number of the TAC class says nothing about the priority with which its jobs are processed. Only the number of processes that you allow for this TAC class specifies how strongly the processing of the jobs is suppressed as compared to other TAC classes.

This method can then be used sensibly when only a few different types of jobs (and therefore only a few TAC classes) in an application and, for example, when you want to prevent long-running jobs from reserving all the processes of an application and therefore unnecessarily slowing down the processing of other important jobs, e.g. administration jobs.



For information on the use of TAC classes in UTM cluster applications, see also the applicable openUTM manual "Using openUTM Applications", section "Using global memory areas" in the chapter "UTM cluster applications".

5.7.3 Comparison of some of the properties of the two methods

You can only use one of the two methods for job control in your UTM application. Which of the two possibilities you should select for your application also depends on the sometimes different properties of the two methods.

Program units with blocking calls

- *Priority control*

Transaction codes from program units that execute blocking calls may be assigned any TAC class as long as a value > 0 is generated in the TASKS-IN-PGWT operand of the MAX statement. You must specify the operand PGWT=YES in the TAC statement for transaction codes with blocking calls.

```
TAC . . . , TACCLASS=number, PGWT=YES
```

This also allows you to process corresponding jobs with different priorities.

- *Process limitation*

All transaction codes from program units that execute blocking calls must be assigned the same dialog or asynchronous TAC class. You must generate these dialog or asynchronous TAC class as follows:

```
TACCLASS . . . , PGWT=YES
```

The corresponding dialog or asynchronous jobs are thus handled in the same way.

Temporarily stopping the execution of certain asynchronous jobs

Both methods for job control provide a mechanism with which you can temporarily prevent the processing of certain asynchronous jobs. These jobs are then received and accepted by openUTM and written in the message queue of the corresponding transaction code. The processing of these jobs is only initiated after the “processing lock” is removed by the UTM administration.

To temporarily prevent the execution of jobs, set the status of the transaction code to KEEP. You can do this during live operation via the UTM administration or do this during the generation of the transaction codes by specifying the following:

```
TAC . . . ,STATUS=KEEP
```

openUTM processes the buffered jobs first if you set the status of the transaction code to ON.



See the openUTM manual “Administering Applications”; KDCADMI operation code KC_MODIFY_OBJECT with object type KC_TAC or the administration command KDCTAC

When using the *process limitation* method, the execution of jobs can also be prevented for all transaction codes of an asynchronous TAC class. In this case you must set the maximum number of processes that are available for jobs of this TAC class to 0.

```
TACCLASS . . . ,TASKS=0
```

openUTM only processes the jobs again if you increase the maximum number of processes.



See the openUTM manual “Administering Applications”; KDCADMI operation code KC_MODIFY_OBJECT with object type KC_TACCLASS or the administration command KDCTCL

You can use both mechanisms, for example, to collect jobs that are to be executed at a later point in time when the load on the application is lower (e.g. at night).



In both cases you should limit the message queue of the transaction code(s) to prevent overloading the page pool with too many buffered jobs. This is done for each TAC by:

```
TAC . . . ,QLEV=
```


Change of process when processing jobs

- *Priority control*

If a service consists of several program units (follow-up TAC after a PEND PA/PR), then a change of process can always occur when processing the service, regardless of whether the current TAC and follow-up TAC belong to the same TAC class or not.

- *Process limitation*

For job control via process limitation, openUTM guarantees that no change of process will occur after a PEND PA/PR and SP when the service TAC and follow-up TAC are assigned the same TAC class.

If the current TAC and follow-up TAC belong to different TAC classes, then a change of process can also occur when using this method.

Change of process for asynchronous services

When a change of process occurs, an asynchronous service is inactive at first and does not reserve a UTM process although it remains open.

You can limit the maximum number of simultaneously open asynchronous services. You must specify the following in the MAX statement to do this:

```
MAX ...,ASYNTASKS=(...,service_number).
```

If *service_number* of open asynchronous services exist, then no new asynchronous job that is ready is started. An interrupted open asynchronous service is selected from the next process that becomes free, and this service is resumed.

5.7.4 Process priorities in BS2000/OSD

B openUTM uses the methods described above for job control to select a job that is to be
B restarted or resumed. Jobs that are currently being processed cannot be influenced with
B these methods.

B You can use the scheduling mechanisms of the BS2000/OSD for prioritizing jobs to
B influence the priority of the active jobs. The RUNPRIO operand of the TAC statement can
B be used for this purpose. With RUNPRIO you assign a transaction code a process priority
B (Run-priority) in the KDCDEF generation. You can influence the speed with which a running
B job is processed with the process priority. A job for a transaction code with a higher process
B priority will be given preference when distributing the CPU resources in comparison to other
B jobs with lower priorities.

B If you have generated a process priority for a transaction code, then openUTM sets the
B BS2000 process priority of the process that is processing a job for this transaction code to
B the value generated in RUNPRIO.
B You can specify a value between 30 (highest priority) and 255 (lowest priority) in RUNPRIO.

B  **TAC statement** on [page 483](#)
B operand RUNPRIO

5.8 Data access control

When you have services that access security-relevant data, it is sensible to restrict access to a limited number of authorized users. openUTM offers two possible methods of data access control which allow you to set different data access authorizations in a UTM application:

- access list concept (service-oriented)
- lock/key code concept (user-oriented)

Both processes use, for the most part, the same generation interfaces.

The greatest difference lies in the way in which the UTM objects are seen: The access list concept allows you to specify a list of codes for each service. These codes specify which user (types) are permitted to access the data. The lock/key code concept allows you to define an (individual) lock code for each service and then assigns each user the appropriate key codes.

Services whose TACs are not secured by a lock code or access list can be called by all users without restriction.



For detailed information about the access list and the lock/key code concepts see the openUTM manual “Concepts und Functions”.

5.8.1 Lock/key code concept

A lock code is a number which symbolizes a logical lock. The objects that are to be protected - for example, the LTERM partner and the transaction codes assigned to the services - are assigned a lock code (TAC or LTERM statement).

Key codes are defined for user IDs and for LTERM partners (USER or LTERM statement). Only when the key code corresponds to the lock code of a protected object is access to this object permitted.

Since a user ID or LTERM partner usually has access to several services, they must also have several key codes. The individual key codes are thus organized into key sets (KSET statement).

The lock/key code concept has the following significance:

- It is only possible for a (UTM) user to sign on or to sign on under a UTM user ID if the specified user ID is assigned a key code which corresponds to the lock code of the LTERM partner via which sign-on is performed.
- A user can only call a service when **both** the key set of the current (UTM) user ID **and** that of the LTERM partner contain a key code that corresponds to the lock code of the transaction code.



KSET statement on [page 337](#)

You can use the following operands to define a key set.

- *keysetname*

Name of the key set.

- KEYS=

When assigning a key set to a user (USER):

Specification of one or more key codes (numeric) that are assigned to the user.

When assigning a key set to an LTERM partner (LTERM):

Specification of one or more key codes (numeric) that are assigned to the LTERM partner.



TAC statement on [page 483](#)

You can use the following operands to control access to the TAC.

- *tacname*

Name of the TAC.

- LOCK=

Specifies the lock code that is assigned as a form of logical combination lock to the TAC of a service.

A service that is protected by a lock code can only then be started if the key set of the user **and** the key set of the LTERM partner both contain a key code that corresponds to the lock code.

This operand may not be specified in conjunction with the operand ACCESS-LIST=.

**USER statement** on [page 530](#)

You can use the following operands to assign a key set to a user.

- *username*
UTM user ID.
- KSET=
Specifies the name of the key set that is assigned to the user ID. The key set must be defined using the KSET statement. A maximum of one key set can be assigned to a user.
A user is only able to access a service whose first TAC is protected by a lock code if one of the key codes in the key set of the user corresponds to the lock code. Otherwise access to the service is denied.

**LTERM statement** on [page 355](#) / **TPOOL statement** on [page 511](#)

You can use the following operands to assign a key set to an LTERM partner.

- *ltermname*
Name of the LTERM partner (only for LTERM statement).
- LTERM= , NUMBER=
Name of the LTERM partner (only for TPOOL statement).
- KSET=
Specifies the name of the key set that is assigned to the LTERM partner. The key set must be defined using the KSET statement. For the LTERM partners of a UPIC client or a TS application without an explicitly generated connection user ID this key set is also the key set of the connection user ID.
- USER-KSET= (only for TPOOL statement)
In LTERM pools for TS applications or UPIC clients this specifies the name of the key set that is assigned to the connection user ID. This key set must be defined using the KSET statement. The access authorizations are derived from the intersection of the key sets from KSET= and USER-KSET=.
- LOCK=
The lock code that is assigned to the LTERM partner as the logical combination lock. Only valid for clients (USAGE=D).
Only a (UTM) user for whom a key set has been generated with a key code that matches the lock code of the LTERM partner can sign on to the application via an access-controlled LTERM partner.

5.8.2 Access list concept

An access list is a number of access codes (numeric codes) that are assigned to a service. The access codes in the access list defines user access to a service and can be interpreted as the roles of the users within the structure of their organization (for example, general users, heads of department, system administrators).

If you use the administration tool WinAdmin you can use meaningful names in place of numeric codes.

An access list is defined using the KSET statement and assigned to a service using the TAC statement. The roles for the user (USER) are also defined and assigned as a key set using a KSET statement. In the same way, it is also possible to assign an LTERM partner a certain number of roles.

A user can only access a service (TAC) protected in this way if both the key set of the user and the key set of the LTERM partner via which the user has signed on contains at least one of the roles that are contained in the access list of the service.



The differences between the lock/key code and the access list concepts are described in detail in the security function section of the openUTM manual “Concepts und Functions”.



KSET statement on [page 337](#)

The following operands can be used to define key sets or access lists.

- *keysetname*

Name of the key set or access list.

- KEYS=

When assigning an access list to a service (TAC):

Specification of one or more roles (as numerical values) that have access to the service protected by *keysetname*.

When assigning a key set to a user (USER):

Specification of one or more roles (as numerical values) that are to be assigned to the user.

When assigning a key set to an LTERM partner (LTERM):

Specification of one or more roles (as numerical values) that may be performed when signing on via this LTERM partner.



When using WinAdmin you may also assign roles with alphanumeric names.

**TAC statement** on [page 483](#)

The following operands are used to control the accesses to the TAC.

- *tacname*
Name of the TAC.
- ACCESS-LIST=
Specifies the access list that controls access to this TAC. Only users whose key set contains at least one of the roles contained in this access list and that sign on via a terminal that has also been assigned one of these roles may access this TAC. ACCESS-LIST may not be specified in conjunction with LOCK.

**USER statement** on [page 530](#)

The following operands are used to assign a key set to a user.

- *username*
UTM user ID.
- KSET=
Specifies the name of the key set that the user ID is assigned to. The key set must be defined using the KSET statement. Each user can be assigned a maximum of one key set.
If a user wishes to access a service that is protected with an access list then at least one of the roles of the user must be contained in the access list. Otherwise access to the service will be denied.

**LTERM statement** on [page 355](#) / **TPOOL statement** on [page 511](#)

The following operands are used to assign a key set to an LTERM partner.

- *ltermname*
Name of the LTERM partner (only for LTERM statement).
- LTERM= , NUMBER=
Name of the LTERM partners (only for TPOOL statement).
- KSET=
Specifies the name of the key set assigned to the LTERM partner. For the LTERM partner of a UPIC client or a TS application without explicitly generated connection user ID this key set is the same as the key set of the connection user ID. The key set must be defined using the KSET statement. Each LTERM partner may be assigned a maximum of one key set.

- USER-KSET= (only for TPOOL statement)

In LTERM pools, specifies the name of the key set for TS applications or UPIC clients that is assigned to the connection user ID. The key set must be defined using the KSET statement. The access authorizations are derived from the intersection of the key sets of KSET= and USER-KSET=.



Access to the LTERM partner may not be protected using access lists. When using access lists to provide data access control to services, you should not use access protection on the LTERM partner, or in other words the parameter LOCK of the LTERM and TPOOL statements may not be specified.

Data access control for service-controlled queues using access lists

It is also possible to protected service-controlled queues from unauthorized read, delete or write access. To do this an access list is defined (TAC/USER statement).



TAC statement on [page 483](#)

The following operands are used to control access for TAC queues.

- *tacname*
- Q-READ-ACL=
- Q-WRITE-ACL=

Name of the access list that controls the read, delete and write access of a user to this queue. The access list must be generated using a KSET statement.

A user only has read or write access to the TAC queue if the key set of the user and the key set of the LTERM partner via which the user has signed on both contain at least one of the roles that are defined in the access list for the TAC queue.

The key set must be generated for the user and the LTERM partner using the USER or LTERM statements.

**USER statement** on [page 530](#)

The following operands can be used to control the access for USER queues.

- *username*
UTM user ID.
- KSET=
Specifies the name of the key set to which the user ID is assigned.
The key set must be defined using the KSET statement. Each user may be assigned a maximum of one key set.
- Q-READ-ACL=
Q-WRITE-ACL=
Name of the access list via which the user is able to protect their own USER queues from read, delete or write access. The access list must be generated using the KSET statement.



The owner of a queue always has read, write and delete authorization for their queue, regardless of whether the read/write authorizations are restricted for other users.

An external user only has read or write access to the USER queue of another user if the key set of the external user and the key set of the LTERM partner via which the external user has signed on each contain at least one of the roles defined in the access list for the USER queue.

If you do not specify Q-READ-ACL/Q-WRITE-ACL all users have read, delete and write authorization within the queue.



For more detailed information on Message Queues see [page 181ff.](#)

5.8.3 Data access control with distributed processing

You can use the data access control mechanisms of openUTM with distributed processing. The protection methods are specified when the applications are generated.

Protection methods in the ordering application

When generating an application you generally initially specify which services of a remote partner application may be called. For each remote service that is to be used you must agree an LTAC local transaction code (LTAC statement). Access is generally denied to remote services for which no LTACs have been agreed.

In order to further graduate the data access control you can also assign lock codes to individual TACs (see [page 219](#)) or use access lists (see [page 222](#)).

A service of the local application can only address a remote service if the service was started under a user ID (KCBENID) and from a client (KCLOGTER) that have the appropriate access permissions.



LTAC statement on [page 348](#)

The following operands are used to define which services of a remote partner application may be called and which access authorizations are placed on the LTAC. The operands ACCESS-LIST and LOCK are mutually exclusive.

- *ltacname*

Name of a local TACs (LTAC) for the remote service program.

- ACCESS-LIST=

Name of an access list. In order to be able to start the remote service program the key set of the user of a local application must have been assigned at least one of the roles defined in the access list (as defined in the USER statement).

The access list must be defined using a KSET statement.

- LOCK=

Definition of the lock code of the remote service program. A service of the local application can only address this remote service if the local service was started under a user ID (KCBENID) and from a client (KCLOGTER) that have the appropriate access permissions.

ACCESS-LIST and LOCK cannot be specified simultaneously.



If you enter neither ACCESS-LIST nor LOCK then the LTAC is not protected and any user of the local application is able to address the remote service program.

Protection measures in the receiving application

In the receiving application (partner/server application) the application sending the job is assigned a key set. Only if this key set contains a key code or access code that corresponds to the lock code or access list of the requested service is it possible for the process requested by the submitting application to be started.

In order to be able to access a remote service, the service that is being called must be generated with a TAC and the following conditions must be fulfilled:

- LU6.1 connections:

The key set of the partner as defined in LPAP ...,KSET= must contain a key code that corresponds to LOCK= or ACCESS-LIST= of the TAC.

- OSI TP connections:

- If a partner attempts to sign on without a user ID, then the key set defined in OSI-LPAP ...KSET and OSI-LPAP ...,ASS-KSET= must contain a code that correspond to LOCK= or ACCESS-LIST= of the TAC.

The access authorizations are derived from the intersection of the key sets of KSET= and ASS-KSET=. Thus KSET= should always be a superset of ASS-KSET=.

You can define suitable restrictions on the key set defined with OSI-LPAP ...,ASS-KSET to ensure that specific TACs cannot be called unless the partner specifies a real user ID.

- If a partner attempts to sign on with a real user ID, then the key set of this user ID and that defined in OSI-LPAP ... KSET= must contain a code that corresponds to LOCK= or ACCESS-LIST= of the TAC.

This also applies to a client/server link with OpenCPIC.



For more detailed information about data access control with distributed processing see openUTM manual “Concepts und Functions”.

5.9 Message encryption on connections to clients

Clients often access UTM services via open networks. This allows unauthorized persons the opportunity to read data from the line and obtain passwords for UTM user IDs or sensible user data, for example. To prevent this, openUTM supports the encryption of passwords and user data on connections to UPIC clients and under BS2000/OSD additionally on connections to certain terminal emulations. Several encryption levels are available for selection for connections to UPIC clients (DES key or AES key with three different key lengths, see [page 231](#)).

Encryption in openUTM not only serves to secure the data on the connection between the client and the server application, but it can also be used to limit access for clients and access to certain services. Two encryption levels are available for selection (DES or AES method, see [page 232](#)).

5.9.1 Requirements

Connecting a server application to a UPIC client

The requirements for encryption between an openUTM server application and a UPIC client are:

- The openUTM-Crypt encryption component must be available on the server in the UTM system code for encryption on connections to UPIC clients.
For legal reasons the encryption functions in openUTM are supplied as a separate product, openUTM-CRYPT, that must be installed separately (for more information see the openUTM Release Notice and for openUTM under BS2000/OSD the installation information in the openUTM manual “Using openUTM Applications under BS2000/OSD”).
- openUTM-Client for the UPIC carrier system with the encryption functions must be used in the UPIC client.

B Connecting a server application to a terminal emulation (BS2000/OSD)

B The encryption of VTSU is offered for connections between UTM applications under
B BS2000/OSD and terminal emulations. VTSU-B uses a separate key management. In this
B manner the encryption of openUTM is not used on connections to a terminal emulation and
B the openUTM-CRYPT product is not required. The data and system access control mecha-
B nisms that come in conjunction with encryption are in effect, however. openUTM receives
B information from VTSU via the encryption level that was negotiated for the connection to the
B client.

- B** The following requirements must be fulfilled:
 - B** ● One requirement is the use of VTSU-B and the VTSU-SEC selectable unit.
B Which of the current versions you must use is described in the release notes for
B openUTM. You can consult the Release Notice for VTSU-SEC to determine which
B VTSU parameters must be set.
 - B** ● A terminal emulation must be in use on the client that supports the encryption functions
B (e.g. DESK2000).
- B** These communication partners are called VTSU partners in the following.

5.9.2 Encryption methods

openUTM uses a combination of the AES method (Advanced Encryption Standard) and RSA method (named after its authors Rivest, Shamir and Adleman) for encryption. The DES method (Data Encryption Standard) can still be used for partners that do not yet support the AES method.

AES and DES methods

User data and passwords on a connection are encrypted with a symmetrical AES key or a DES key. The client and UTM application both use the same AES/DES key to encode and decode the messages.

The AES/DES key is created by the client and passed to the UTM application when the connection is established. The key is connection-specific, i.e. a separate key is created for every connection, and only this one connection uses this key.

RSA methods

The AES or DES key itself is encrypted before transmission to increase the security. openUTM creates an RSA key pair during generation for this purpose. The RSA key pair consists of a public and a secret, private key. The RSA key is connection-specific and is used to encrypt the AES/DES key on all relevant connections of the UTM application to clients:

- The RSA public key is passed directly to the client when the connection is established from the UTM application.
- The client then encrypts the AES/DES key and transmits the AES/DES key to the UTM application.
- The UTM application decrypts the AES/DES key using the associated private RSA key.

Several encryption levels are available for selection. They differ in the length of the RSA keys used. Consequently, there may be several RSA key pairs with different key lengths in a UTM application.

The administrator of the application can create and activate new RSA key pairs at any time (see [section “Creating the RSA key pair and reading the public key” on page 233](#)).

RSA key pairs are transferred by KDCUPD from an old KDCFILE to the new KDCFILE.

B Encryption methods for BS2000 terminal emulations

B The RSA key pair of the UTM application is not used for connections to VTSU partners. A
B key pair created by VTSU-B is used here. VTSU-B uses the same algorithms and methods
B as openUTM for encryption.

5.9.3 Encrypting passwords and user data

User data and passwords are not passed in encrypted form on connections between UTM application and trusted clients (i.e. clients generated trusted clients; see [point 3 on page 232](#)).

Passwords from (non-trusted) UPIC clients are always encrypted and then passed to the UTM application in openUTM if the client as well as the server supports encryption. Passwords are also encrypted in this case if no encryption was agreed to for the connection.

B Passwords are only passed in encrypted form on connections between UTM applications
B under BS2000/OSD and VTSU partners if encryption was agreed to for the connection or if
B the password was entered in a blanked-out field.

The encryption of **user data** is optional. This is negotiated between the client and the server when a UPIC conversation or connection to a VTSU partner is established.

- The client can force encryption.

The ENCRYPTION-LEVEL keyword in the Side Information file and the *Set_Encryption_Level* function call are available for a UPIC client for this purpose.

B The encryption level is defined on the host for VTSU partners. Various encryption levels
B can be specified, from unconditional encryption for all applications through the
B encryption of individual messages that the user himself has selected.

- A UTM application can request encryption for a certain service or a certain partner.

If one of the partners requests encryption, then the request for encryption is either accepted by the other side or the conversation/connection between the partners is not established.

Encryption is always negotiated on a conversation-to-conversation or connection-to-connection basis. Message-specific encryption via the program interface is not possible.

You can assign every client and every service an encryption level in the configuration of the UTM application. The encryption level specifies whether or not messages from the client must be encrypted. The encryption levels are defined with the KDCDEF option ENCRYPTION-LEVEL in the TAC, PTERM and TPOOL statements.

The encryption levels can be used by openUTM to control the access of clients as well as the access to certain services.

5.9.3.1 System access control

You can specify an encryption level for every client (PTERM) and every client group (LTERM pool; TPOOL) in the UTM configuration. The encryption level specifies if and how clients must encrypt messages or may encrypt messages. In this manner a UTM application can protect itself from accesses via insecure clients.

You specify the encryption level for a client in the KDCDEF generation in the PTERM statement of the client:

```
PTERM . . . ,ENCRYPTION-LEVEL=
```

You specify the encryption level as follows for clients that connect to the application via an LTERM pool:

```
TPOOL . . . ,ENCRYPTION-LEVEL=
```

There are following encryption levels:

1. openUTM requests the use of encryption from the client.
The client must encrypt in all cases, otherwise it will not gain access to the UTM application. The minimum length of the RSA key used is predefined. If the partner does not support encryption or cannot use the RSA key of the requisite key length, then it cannot establish any connections to the UTM application.

In this case, generate with following variants:

```
ENCRYPTION-LEVEL=1 (RSA key length 200 byte, DES methods)
ENCRYPTION-LEVEL=2 (RSA key length 512 byte, AES methods )
ENCRYPTION-LEVEL=3 (RSA key length 1024 byte, AES methods )
ENCRYPTION-LEVEL=4 (RSA key length 2048 byte, AES methods )
```

2. openUTM does not request encryption and the client can specify whether or not the connection is to use encryption.

The client is also allowed access without encryption, but it must encrypt if a service explicitly demands it (see [section "Data access control" on page 232](#)).

In this case, generate with:

```
ENCRYPTION-LEVEL=NONE
```

3. The client is trusted (*trusted client*). Encryption is not used on connections to such clients. A trusted client can also call „protected“ services without encryption (see section below).

You should only generate clients as trusted when you are sure that communication occurs via a secure line.

In this case, generate with:

```
ENCRYPTION-LEVEL=TRUSTED
```

X/W
X/W



Unix systems, Windows:

Local UPIC clients (type UPIC-L) are always trusted clients.

5.9.3.2 Data access control

You can protect individual services from accesses via insecure clients with the help of the encryption functions. A client may only access "protected" services if it is a trusted client or if it is able to encrypt using the requisite method.

You can protect a service by assigning encryption level 1 or 2 to the corresponding service TAC:

```
TAC . . . ,ENCRYPTION-LEVEL=1 (encryption according to the DES method)
```

```
TAC . . . ,ENCRYPTION-LEVEL=2 (encryption according to the DES method)
```

If a service is protected in this manner, then the following is true:

- A trusted client can start such a service without using encryption.
- For non-trusted clients that support the encryption function, the service belonging to the transaction code is only started if the client has passed the input message encrypted with the requisite method. Otherwise
 - In the case of UPIC clients, conversation establishment is rejected by openUTM.
 - In the case of VTSU partners, this leads to a BADTAC or message K009 is output.

B

If the service is called via a transaction code without user data (e.g. for terminal emulations via a function key) or started due to service chaining, then the service is also started without encryption. openUTM encrypts then all dialog output messages to the client. openUTM expects all further input messages from the client to be encrypted for multi-step services. If the input message contains unencrypted user data, then the service is terminated abnormally.

- If a non-trusted client that does not support the encryption functions attempts to start the service, then the conversation to the UPIC client is rejected. If an attempt is made to start the service using service chaining, the service is terminated abnormally. If the service is called via a VTSU partner (terminal emulation), then this leads to a BADTAC or message K009is output.

B
B

Encryption is optional when you generate a service TAC as follows (default):

```
TAC . . . , ENCRYPTION-LEVEL=NONE
```

Information for encryption on the KDCS program interface

You also have the possibility of writing separate program units that execute an access authorization check. Encryption data is displayed on the program interface for the INIT PU call. The following information is displayed:

- the encryption levels that are generated for the client and transaction code
- whether encryption was negotiated for the conversation
- whether the client supports encryption in principle
- whether the last input message was encrypted

5.9.4 Creating the RSA key pair and reading the public key

You should replace the RSA key pair with a new RSA key pair in your UTM application in regular intervals for security reasons. The administration program interface and the administration tool openUTM-WinAdmin provide the corresponding functions.



See the openUTM manual “Administering Applications”; KDCADMI operation code 4KC_ENCRYPT or the help system for openUTM-WinAdmin.

With the help of the administration you can create a new key pair, read the public key and activate the new key pair. Only after activation can the new key pair be used by the UTM application for encryption. An activated key pair can also be deleted using administration facilities.

To further increase the security of the data on a connection you can read the public key of the RSA key pair, pass it to the client using your own method and store it there. You should only activate the new RSA key pair once this has been accomplished. With the help of the public RSA key you have stored, the client can verify if the public key received over the connection to the UTM application really came from the UTM application.

5.10 Defining database linking

When configuring the application you must use the KDCDEF control statements to define the database system with which the UTM application is to coordinate.



If a UTM application is to be linked with a database, additional parameters must be specified when linking and starting. See also the openUTM manual “Using openUTM Applications”.

The remaining UTM generation is not affected by the linking.

5.10.1 Linking databases under BS2000/OSD

B openUTM supports coordination with the following database systems:

- B** – UDS/SQL
- B** – SESAM/SQL
- B** – XA
- B** – PRISMA
- B** – LEASY (the LEASY file systems behaves like a database system in relation to openUTM)

B A UTM application can work in coordination with up to 2 (up to 8 with special release) different databases. Each database system is defined with a DATABASE statement for the KDCDEF run.

B  **DATABASE statement** on [page 321](#):

B Definition of the database with which the UTM application works together:

B ● ENTRY=

B Entry name of the supported database, which can be seen in the table on [page 321](#).

B ● LIB=

B Object module library from which the connection module to the database system is to be loaded dynamically.

B ● TYPE=

B Type identifier of the database system.

B – You can connect to database systems not contained in the list above but that support the IUTMDB interface with TYPE=DB.

B – The link to a XA resource is generated with TYPE=XA.

5.10.2 Linking to a Resource Manager under Unix systems and Windows systems

X/W openUTM is linked with Resource Managers (e.g. database systems) via the XA interface
 X/W standardized by X/Open. It coordinates the transactions of openUTM with the services of
 X/W the Resource Manager. The XA interface is supported in the CAE version of the XA
 X/W interface (XA-CAE).

X/W openUTM for Unix systems and Windows systems supports coordination with the following
 X/W data base systems:

- X/W – Oracle
- X/W – INFORMIX

X/W  **RMXA statement** on [page 466](#)

X/W The Resource Manager to which openUTM is to be linked and the version of the
 X/W XA interface via which the link is to be made must be defined in the generation with
 X/W the RMXA statement:

- X/W ● XASWITCH=

X/W Name of the xa_switch_t structure of the Resource Manager, which is made known
 X/W to openUTM.

X/W The following must be noted for the linked operation of openUTM with XA:

- X/W ● Several Resource Managers (i.e. database systems) can be served within a UTM appli-
 X/W cation.
- X/W ● It is not permitted to generate more than one Resource Manager with the same gener-
 X/W ation parameters. In other words, within a generation run there cannot be more than one
 X/W RMXA statement with the same name for the xa_switch_t structure of the Resource
 X/W Manager.

- The simultaneous operation of several entities (databases) of a Resource Manager (database system) is possible provided the Resource Manager supports multi-instance mode. The databases with which the UTM application is linked are determined by corresponding start parameters for the application. For multi-instance mode, you must specify several open strings at the start.

Below is a description of how you must generate the linking of your UTM application with the individual Resource Managers for Oracle and INFORMIX. The database-specific names specified here (xa_switch_t structure) may change, which is why you should check that the specifications are correct. For more information, see the documentation for the individual database systems.

Linking with Oracle

On Windows systems, only the static XA link is supported.

RMXA XASWITCH=xaosw

or.

RMXA XASWITCH=xaoswd

With SPEC=C you specify that linking is via the XA-CAE version of the XA interface.

Linking with INFORMIX

RMXA XASWITCH=inf_xa_switch

5.11 Internationalizing the application – XHCS support (BS2000/OSD)

B A UTM application in BS2000/OSD can be programmed such that communication partners
B with different languages can receive the messages from the program units in their
B respective language. Even regional differences within a language can be taken into
B account. Date specifications, time, units of measurement, and currency symbols can be
B displayed in accordance with language-specific conventions.

B To display the fonts and special characters of the individual languages on a terminal or
B printer, you may require various extended character sets (8-bit codes). Using the BS2000
B software product XHCS (**Extended Host Code Support**), several extended character sets
B can be used simultaneously in a BS2000/OSD system. openUTM supports the functions of
B XHCS. This means that you can assign a particular language environment – also called a
B locale – to the UTM objects. In other words, you can assign a standard locale. Individual
B users and LTERM partners that clients use to connect to the application are assigned
B specific locales that are used to edit the messages.

B To implement multilingualism in UTM applications, openUTM offers the following functions.

- B ● When generating the application, specific languages and the character sets to be used
B for output can be assigned to the application, the user IDs, the LTERM partners, and
B the LTERM partners of the LTERM pools. In this case you define locales, which define
B the language environment and character set, for the objects.
- B ● You can define locales for user message modules that take account of language-
B specific requirements. These language-specific message modules are assigned to
B users and LTERM partners whose language and territorial identifiers match the locale
B of the language-specific message module. See also [section “UTM messages” on
B page 186](#).
- B ● While the application is running, you can change the assignment of language and
B character set for your user ID. The KDCS interface provides the SIGN CL call for this
B purpose.
- B ● Using the variants INIT PU and INFO LO of the function calls INIT and INFO, a UTM
B program can read the language and character set of the user ID, the application, a
B particular LTERM partner, or the LTERM partners in a pool. The program unit thus
B obtains information on the character sets supported by the terminal and the character
B set of the input message. With this information, the program unit can correctly interpret
B the input of the user and send messages to the user in the correct language and with
B the appropriate character set.

- B ● If the message of a program unit is sent to a terminal/printer, openUTM transfers the logical message of the program unit to VTSU-B together with the name of the character set to be used for editing. VTSU-B edits the message for outputting to the terminal or printer. For information on the character set is used to edit a message please refer to the openUTM manual "Programming Applications with KDCS".
- B If the job submitter in a service is a partner program, the logical message is transferred to the job submitter without editing.
- B ● The program unit can use INFO LO to request information from openUTM regarding the language and character set of the LTERM partner, and the character sets supported by the terminal/printer assigned to this LTERM partner. The character set used to edit the message for outputting to the terminal/printer must be compatible with one of the character sets supported by this terminal/printer.
- B Before discussing these functions, we will explain specific XHCS terms.

5.11.1 Definitions of XHCS terms

B ISO character sets, variant numbers

- B Various extended character sets for various language areas are standardized in ISO 8859, for example ISO 8859-1, ISO 8859-2, etc. The numbers at the end (-1, -2, etc.) are called the *variant numbers*. An extended character set contains all the characters required to represent the language of a language area.
- B ISO 8859 codes are extensions of the ASCII code ISO 646. They are used by terminals and Unix systems, for example. All ISO 8859 character sets contain the ASCII code as the shared part in the low-order half of the code table.

B EBCDIC character sets

- B EBCDIC character sets are used in the BS2000 operating system. An extension of EBCDIC.DF.03-IRV or -DRV exists for each ISO 8859 code. EBCDIC.DF.03-IRV is the international reference version and EBCDIC.DF.03-DRV is the German reference version of the non-extended EBCDIC code. Both codes contain the EBCDIC kernel as the shared character set and only differ in certain symbols. The extensions of these EBCDIC character sets are called EBCDIC.DF.04-1, EBCDIC.DF.04-2 through EBCDIC.DF.04-10.

B Compatible character sets

B Extended ISO and EBCDIC character sets with the same variant number are *compatible*, i.e. they contain the same characters. The individual characters are located at different code positions within the code table. The codes can be transferred using conversion tables.

B The BS2000 system administrator can use XHCS to modify the EBCDIC character sets by assigning different code positions in the code table to the individual characters of a character set. The complete set of characters is retained. The modified EBCDIC character sets are *compatible* with the EBCDIC.DF.04-n character set from which they were generated.

B Reference code

B XHCS combines all compatible character sets of the system into a group. A group therefore contains an ISO variant and the EBCDIC character sets compatible with this variant. The EBCDIC.DF.04-n character set of the group is the reference code of the group. All character sets in a group can be converted to the reference code of the group using XHCS.

B Coded character set name (CCS name)

B A name containing a maximum of eight characters, known as the CCS name or the CCSN, is assigned to each character set used in the system. The CCS name uniquely identifies the character set in the system. The CCS names of the reference codes are predefined by XHCS. EBCDIC.DF.04-1 has the CCS name EDF041, for example.

B A list of the CCS names for the character sets available in your BS2000 can be obtained using EDT Version 16.4 or later. To request this information, call EDT and enter the EDT statement @SHOW CCS. EDT then supplies a list of the available character sets.


B Default system code

B The BS2000 system administrator can define several extended character sets (also for various ISO variants), which can be used simultaneously by the system components.

B The system administrator can define one of these character sets as the default system code. The default system code currently set is indicated in the output of the command /SHOW-SYSTEM-PARAMETERS PAR=*AL. It is specified in the HOSTCODE parameter.

B **Default user character set**

B The BS2000 system administrator can assign one of the character sets defined in the
B system as the default user character set for each BS2000 user ID. If a default user character
B set is defined for the BS2000 user ID, its CCS name is displayed in the output field CODED-
B CHARACTER-SET of the /SHOW-USER-ATTRIBUTES command.

B  For further information on XHCS, see the User Guide “XHCS 8-Bit Code Processing
B in BS2000/OSD - Internationalization”.

5.11.2 Defining the language environment – setting the locale

B When generating a UTM application, a separate language environment can be defined for
B the UTM application, for each LTERM partner, for all LTERM partners in an LTERM pool,
B and for each user ID. To do this, you assign the application and the individual objects a
B triplet comprising the language identifier, territorial identifier, and name of a character set,
B which is known as the locale. The locale is specified as follows:

B LOCALE=(lang_id,terr_id,ccsname)

B *lang_id* The language identifier *lang_id* identifies the language in which the user is
B to be addressed by the UTM program units. The language identifier can be
B up to 2 bytes long. The descriptor of a language can be freely selected.


B *terr_id* The territorial identifier *terr_id* enables you to take account of regional differ-
B ences within a language (e.g. English in England and America) or different
B units of currency and measurement in the various countries (dollar and
B sterling). The territorial identifier can be up to 2 bytes long and can be freely
B selected.

B *ccsname* The character set name *ccsname* specifies which character set can be used
B to edit a message for outputting to the terminal. As the character set name,
B specify the CCS name of a character set defined in the BS2000 system.
B CCS names are assigned by the BS2000 system administrator.

B If all users come from the same language area, e.g. Western Europe, it is sufficient to
B assign an extended character set to the UTM application. It is only necessary to use user-
B specific character sets if the various users of an application speak languages that cannot
B all be represented by an extended character set.

B In order to support extended character sets, the subsystem XHCS must be available on the
B processor on which the UTM application is running. For all character set names generated
B in the UTM application, associated EBCDIC character sets must be defined in XHCS. In
B addition, the terminals must support an ISO character compatible with the respective
B EBCDIC character set. Only particular types of terminals and printers support 8-bit
B character sets.


B Application-specific language environment – standard-language environment

B  **MAX statement** on [page 368](#)
 B You assign the locale to the UTM application in the generation using the MAX
 B statement:

- B ● LOCALE=

B The locale generated for the application is assigned to each user ID, each LTERM
 B partner, and each LTERM pool as the default value for the language environment.
 B This default setting applies as long as no specific locale is defined for these objects.


B User-specific language environment

B  **USER statement** on [page 530](#)
 B You assign a locale to a user ID using the USER statement:

- B ● LOCALE=

B The character set assigned to a user ID is used to output dialog messages to the
 B screen (see the character sets section of the openUTM manual “Programming
 B Applications with KDCS”).

B LTERM partner-specific language environment

B  **LTERM statement** on [page 355](#) and **TPOOL statement** on [page 511](#)
 B You use the LTERM statement to assign a locale to an LTERM partner via which a
 B terminal or printer connects to the application. For an LTERM pool, a locale is
 B defined for all LTERM partners in this pool using the TPOOL statement:

- B ● LOCALE=

B The character set defined for the LTERM partner is used to output asynchronous
 B messages (see also [page 186](#)).

B The LTERM partner-specific locale is also used in the first part of the sign-on
 B service, for example, if the user has not yet signed on, i.e. the user-specific
 B language environment is not yet created.

B Example

B The language identifier DE for German is used in the application. To be able to take account
B of the different units of currency in messages to users in Germany and Switzerland (Euro
B and franc), the territorial identifiers `De` for Germany and `CH` for Switzerland are defined. The
B EBCDIC character set EBCDIC.DF.04-1 can be used to output messages. Its CCS name is
B EDF041.

B ● The locale for users in Germany can be defined as the standard language environment
B for the application. To this end, specify the following in the MAX statement:

B `MAX . . . , LOCALE=(DE,DE,EDF041)`

B In this case, no separate locale need be defined for users and terminals in Germany
B that connect to the application via LTERM partners.

B ● If language-specific requirements are to be taken into account for users and terminals
B in Switzerland, the following must be generated:

B `USER username , . . . , LOCALE=(DE,CH,EDF041)`
B `LTERM ltermname , . . . , LOCALE=(DE,CH,EDF041)`

B ● However, you can also use the DEFAULT statement to set the locale (DE,DE,EDF041)
B for all USER and LTERM statements:

B `DEFAULT USER LOCALE=(DE,DE,EDF041)`
B `DEFAULT LTERM LOCALE=(DE,DE,EDF041)`

B You must then also generate the following for users and terminals in Switzerland that
B connect to the application via LTERM partners:

B `USER username , . . . , LOCALE=(,CH)`
B `LTERM ltermname , . . . , LOCALE=(,CH)`

5.11.3 Character set names for edit profiles and formats

B In addition to the user-specific and LTERM partner-specific assignment of character set
B names, a separate character set name can be assigned to each edit profile defined in the
B application.

B The name of a character set can be assigned to each format when creating formats with
B FHS/IFG.

B For information on which of the generated character set names (application-specific, user-
B specific, LTERM partner-specific character set, or the character set name assigned to an
B edit profile or to a format) is used to edit a message for outputting to the screen or printer,
B as described in the openUTM manual “Programming Applications with KDCS”.

5.12 Generating system access control using Kerberos (BS2000/OSD)

B The following generation statements are of significance for generating access control using
B the distributed authentication service Kerberos:

B ● LTERM KERBEROS-DIALOG=

B If you specify LTERM KERBEROS-DIALOG=YES, a Kerberos dialog is carried out
B when a connection is established for terminals that support Kerberos and that connect
B to the application directly via this LTERM partner (not via OMNIS) (see [page 359ff](#)).

B ● TPOOL KERBEROS-DIALOG=

B If you specify TPOOL KERBEROS-DIALOG=YES, a Kerberos dialog is carried out
B when a connection is established for terminals that support Kerberos and that connect
B to the application directly via this terminal pool (not via OMNIS) (see [page 518ff](#)).

B openUTM stores the Kerberos information in the length resulting from the maximum lengths
B generated for MAX PRINCIPAL-LTH and MAX CARDLTH (see [page 392](#) and [page 376](#)). If
B the Kerberos information is longer, it is truncated to this length and stored.

6 The KDCDEF generation tool

In order to generate a new UTM application or adapt an existing UTM application, you must first define the application configuration using KDCDEF control statements, and then use the KDCDEF generation tool to generate the UTM components KDCFILE and KDCROOT from which the UTM application program is created. For further information, see [chapter “Introduction to the generation procedure” on page 31](#).

You can also modify the configuration of an application dynamically during operation. To do this, the RESERVE statement can be used during generation to reserve certain table locations for UTM objects. You can thus insert or remove clients, printers, user IDs, and services, KSETs, LTACs, CONs and LSEs in the configuration “on-the-fly” without affecting availability. The dynamic entry of objects is described in detail in the openUTM manual “Administering Applications”.

By issuing a CREATE-CONTROL-STATEMENTS statement during the KDCDEF run, you can read out the configuration information defined in the KDCFILE of a dynamically configured application, and convert this information to control statements. This function is known as inverse KDCDEF. The control statements which are generated in this way are written to a file which you can re-use directly as the input file for the KDCDEF run. For more information, see [section “Inverse KDCDEF” on page 268](#).

6.1 Creating the ROOT table source and KDCFILE

Based on the configuration information in the KDCDEF control statements, the KDCDEF generation tool creates the KDCFILE. This file contains all configuration and administrative data, as well as the ROOT table source for the main routine KDCROOT.

The KDCFILE and the ROOT table source can be generated simultaneously in a single KDCDEF run or individually in separate KDCDEF runs. This is defined in the KDCDEF statement OPTION ...,GEN=.

All KDCDEF statements provided for defining the UTM application are listed in the following sections in accordance with their function group.

6.1.1 Statements for controlling the KDCDEF run

Statement	Function
EJECT	Initiate a page feed in the log
END	Terminate KDCDEF input
OPTION	Manage the KDCDEF run
REMARK or *	Insert a comment line
<i>additional statement under BS2000/OSD</i>	
B DEFAULT	Define default values

6.1.2 Statements for creating the ROOT table source

Statement	Function
AREA	Define names for additional data areas
EXIT	Define event exits
FORMSYS	Define the format handling system
MAX	Define UTM application parameters
MESSAGE	Define the UTM message module
PROGRAM	Define program units
RESERVE	Reserve table locations for objects that can be entered dynamically
ROOT	Define a name for the ROOT table source
<i>additional statements under BS2000/OSD</i>	
B DATABASE	Define the database system (BS2000/OSD)
B LOAD-MODULE	Define load modules for BLS
B MPOOL	Define a common memory pool
B TCBENTRY	Define a group of TCB entries
<i>additional statements under Unix systems and Windows systems</i>	
X/W X/W RMXA	Define a name for a resource manager on Unix systems and Windows systems (database connection via the X/Open XA interface)
X/W SHARED-OBJECT	Define shared objects/DLLs for exchanging programs

6.1.3 Basic statements for creating a KDCFILE

Statement	Function	
ACCOUNT	Define UTM accounting parameters	
BCAMAPPL	Define additional application names for parallel connections	
CLUSTER	Define global properties of a UTM cluster application	
CLUSTER-NODE	Define a node application of a UTM cluster	
CREATE-CONTROL-STATEMENTS	Create control statements from the existing KDCFILE for a new KDCDEF run	
KSET	Define a key set	
LTERM	Define an LTERM partner as the logical access point for clients and printers	
MAX	Define the name and runtime parameters of the UTM application	
MESSAGE	Define a UTM message module	
MSG-DEST	Define a user message line	
PROGRAM	Define the names and properties of program units	
PTERM	Define clients and printers	
QUEUE	Reserve table entries for temporary message queues	
RESERVE	Reserve table locations for objects that can be entered dynamically	
SFUNC	Define special functions for the F and K keys	
SIGNON	Control the sign-on procedure	
TAC	Define the names and properties of transaction codes	
TACCLASS	Define the number of processes for a TAC class	
TAC-PRIORITIES	Define priorities for the TAC classes	
TLS	Define a name for a TLS block	
TPOOL	Define an LTERM pool	
ULS	Define the names of ULS blocks	
USER	Define user IDs	
B	<i>additional basic statements under BS2000/OSD</i>	
B	DATABASE	Define the database system
B	EDIT	Define edit options
B	LOAD-MODULE	Define load modules for BLS
B	MPOOL	Define a common memory pool
B	MUX	Define a multiplex connection
B	SATSEL	Define SAT logging

	Statement	Function
X/W		<i>additional basic statement under Unix systems and Windows systems</i>
X/W	SHARED-OBJECT	Define shared objects/DLLs for exchanging programs

6.1.3.1 Creating the KDCFILE - additional statements for distributed processing via LU6.1

Statement ¹	Function
BCAMAPPL	Define additional application names for parallel connections
CON	Define a logical connection to a UTM partner application
LPAP	Define an LPAP partner as the logical access point for a UTM partner application
LSES	Define a session name for the connection between two UTM applications
LTAC	Define local names for TACs in UTM partner applications
MASTER-LU61-LPAP	Define the master LPAP of an LU6.1-LPAP bundle
SESCHA	Define the session characteristics
UTMD	Define the global values

¹ The basic statement RESERVE can also be used for CON, LSES and LTAC

6.1.3.2 Creating the KDCFILE - additional statements for distributed processing via OSI TP

Statement ¹	Function
ABSTRACT-SYNTAX	Define the abstract syntax
ACCESS-POINT	Create an OSI TP access point for the local UTM application
APPLICATION-CONTEXT	Define the application context
LTAC	Define local names for TACs in UTM partner applications
MASTER-OSI-LPAP	Define a master LPAP for a OSI-LPAP bundle
OSI-CON	Define a logical connection to the partner application
OSI-LPAP	Define an OSI-LPAP partner as the logical access point for the partner application
TRANSFER-SYNTAX	Define the transfer syntax
UTMD	Define global values and the address of the local UTM application
<i>additional statements under Unix systems and Windows systems</i>	
X/W MAX XAPTPSHMKEY	Define the key for the XAPTP shared memory segment
X/W MAX OSISHMKEY	Define an authorization key for the OSS shared memory segment
X/W MAX OSI-SCRATCH-AREA	Define the size of the working area for dynamic data storage

¹ The basic statement RESERVE can also be used for LTAC

6.1.3.3 Creating the KDCFILE - additional statements for UTM cluster applications

Statement	Function
CLUSTER	Define global properties of a UTM cluster application
CLUSTER-NODE	Define a node application of a UTM cluster application

6.1.3.4 Statements used to generate the UTM cluster files

Statement	Function
CLUSTER ¹	Define global properties of a UTM cluster application
CLUSTER-NODE	Define a node application of a UTM cluster application
MAX ² APPLIMODE ,APPLINAME ,GSSBS ,KB ,LSSBS ,NB, ,ULS ,VGMSIZE	Define UTM application parameters
OPTION ,GEN=(CLUSTER,..	Generate the UTM cluster files

¹ An exception here is an increase in the size of the cluster page pool (operand PGPOOL=(number,...) without modifying the number of cluster page pool files.

² If the values of the operands listed here are modified then the UTM cluster files must be regenerated with OPTION GEN=(CLUSTER,...).

6.1.4 Effects of the KDCDEF statements on the generation objects

Not all statements of the KDCDEF generation tool have the same effect on the KDCFILE and ROOT table source. The table below shows which control statements affect which generation objects during the KDCDEF run:

KDCDEF control statement	KDCFILE	ROOT tables	KDCDEF control	Distributed processing via	
				LU6.1	OSI TP
ABSTRACT-SYNTAX	X				X
ACCESS-POINT	X				X
ACCOUNT	X				
APPLICATION-CONTEXT	X				X
AREA	X	X			
BCAMAPPL	X			X	
CLUSTER	X				
CLUSTER-NODE	X				
CON	X			X	
CREATE-CONTROL-STATEMENTS ¹					
EJECT			X		
END			X		
EXIT	X	X			
FORMSYS ²	X	X			
KSET	X				
LPAP	X			X	
LSES	X			X	
LTAC	X			X	X
LTERM	X				
MASTER-LU61-LPAP	X			X	
MASTER-OSI-LPAP	X				X
MAX ³	X	X			X
MESSAGE	X	X			
MSG-DEST	X				
OPTION ⁴	X	X	X	(X)	(X)
OSI-CON	X				X
OSI-LPAP	X				X

	KDCDEF control statement	KDCFILE	ROOT tables	KDCDEF control	Distributed processing via	
					LU6.1	OSI TP
	PROGRAM	X	X ⁵			
	PTERM	X				
	QUEUE	X				
	REMARK			X		
	RESERVE	X	X ⁶			
	ROOT		X			
	SESCHA	X			X	
	SFUNC	X				
	SIGNON	X				
	TAC	X				
	TACCLASS	X				
	TAC-PRIORITIES	X				
	TLS	X				
	TPOOL	X				
	TRANSFER-SYNTAX	X				X
	ULS	X				
	USER	X				
	UTMD	X			X	X
B	<i>BS2000/OSD specific statements</i>					
B	DATABASE	X	X			
B	DEFAULT ⁷	X	X	X		
B	EDIT	X				
B	LOAD-MODULE	X	X ⁸			
B	MPOOL	X	X ⁹			
B	MUX	X				
B	SATSEL	X				
B	TCBENTRY		X			
X/W	<i>Unix system and Windows system specific statements</i>					
X/W	RMXA		X			
X/W	SHARED-OBJECT	X	X			

- ¹ Based on the configuration information defined in an existing KDCFILE, the CREATE-CONTROL-STATEMENTS statement generates an input file containing KDCDEF control statements for a new KDCDEF run.
- ² Under Windows systems, the statement has no effect, no formatting system is supported.
- ³ The operands CLRCH=, KB=, NB= and SPAB= only affect the generation of the ROOT table source. The other operands only affects the generation of the KDCFILE.
- ⁴ The effect of the OPTION statement on the KDCFILE and the ROOT table source depends on the values entered for OPTION ...,GEN=.
- ⁵ Only when generating a UTM application **without** load modules (under BS2000/OSD), shared objects (under Unix systems) or DLLs (under Windows systems).
- ⁶ Only when generating without the operand PROGRAM= and without load modules, shared objects or DLLs.
- ⁷ The effect of the DEFAULT statement on the KDCFILE and the ROOT table source depends on the specified substatement.
- ⁸ Only when extending the generation by *n* load modules.
- ⁹ Only when generating without load modules.

The MAX, ULS, CLUSTER and CLUSTER-NODE statements also affect the UTM cluster files. If you change parameters of the ULS, CLUSTER and/or CLUSTER-NODE statement when performing a new generation, you must specify OPTION GEN=CLUSTER in order for the changes to take effect, see also [page 417](#).

The KDCDEF control statement OPTION...GEN= is used to define which objects (the KDCFILE, ROOT table sources and UTM cluster files) are to be generated by the KDCDEF generation tool.

When a new ROOT table source is created, this must be compiled (assembled under BS2000/OSD) and relinked to your application. Relinking of an application program is only necessary if the table module is not dynamically loaded. This is not necessary if you merely modify the KDCFILE. You can run the application with the new KDCFILE and the old main routine KDCROOT.

6.2 Calling KDCDEF and entering the control statements

6.2.1 Starting KDCDEF and executing a KDCDEF run



You can also start the KDCDEF run from WinAdmin. For further information, please see the openUTM WinAdmin online Help system.

6.2.1.1 BS2000/OSD

B The KDCDEF generation tool is started using the command:

```
B START-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM
B           (LIBRARY=$userid.SYSLNK.UTM.061.UTIL,ELEMENT-OR-SYMBOL=KDCDEF)
```

B By default, KDCDEF reads the KDCDEF control statements from SYSDTA or from a file.
B The control options for the KDCDEF run (see the OPTION statement, [page 416](#)) are only
B processed by KDCDEF if it is read from SYSDTA. All other control statements for KDCDEF
B can be read from SYSDTA as well as from SAM or ISAM files.

B The SAM or ISAM files can be defined as input sources as described below:

B ● Assign an input file using the BS2000 commando ASSIGN-SYSDTA:

```
B /ASSIGN-SYSDTA TO-FILE=inputsource
B /START-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM
B           (LIBRARY=SYSLNK.UTM.061.UTIL,ELEMENT=KDCDEF)
```

B ● Assign input files using the KDCDEF control statement OPTION ...,DATA=:

```
B /ASSIGN-SYSDTA TO-FILE=*SYSCMD
B /START-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM
B           (LIBRARY=SYSLNK.UTM.061.UTIL,ELEMENT=KDCDEF)
B OPTION DATA=inputsource1
B OPTION DATA=inputsource2
B etc.
B END
```

6.2.1.2 Unix systems

X Proceed as follows to start the KDCDEF tool and to execute a KDCDEF generation:

X 1. Add the directory below to the PATH environment variable:

X *utm-path/ex*.

X The *kdcdef* program used to start the KDCDEF generation tool is located in this directory.

X 2. Create one or more source files with an ASCII editor with control statements for the UTM generation. You must observe the information stated in [section “Order of the control statements” on page 257](#) and [section “Format of the control statements” on page 258](#).

X 3. Create the *filebase* directory (base directory of the application) in which openUTM stores the KDCFILE and other application-specific files. Enter the following command to create this directory:

X `mkdir filebase`

X You must create the directory **before** starting KDCDEF. *filebase* is the directory that you specified in the MAX statement in the FILEBASE= operand.

X 4. You start the KDCDEF tool with the *kdcdef* program.

X By default, KDCDEF reads the KDCDEF control statements from *stdin*. Only the control options for the KDCDEF run are read in from a shell script (see OPTION statement on [page 416](#)), while the actual generation statements for KDCDEF are read from the files created in Step 2. You can specify these files directly at the start of KDCDEF:

X `kdcdef < definput`

X or after KDCDEF has been started using the KDCDEF statement OPTION:

X `OPTION DATA=definput`

X `END`

X The messages and logs from KDCDEF are written to *stdout* and *stderr*, i.e. everything is displayed on the screen if you have not redirected the output. You can redirect the output to a file as follows (you can select any name you want for the files):

X `kdcdef < definput 2>def.err 1>def.prot`

X All UTM messages are recorded in *def.err* and *def.prot* contains the complete log of the KDCDEF run.

6.2.1.3 Windows systems

W Proceed as follows to start the KDCDEF tool and to execute a KDCDEF generation:

W 1. Add the directory below to the PATH environment variable:

W *utmpath*\ex.

W The *kdcdef.exe* program used to start the KDCDEF generation tool and other utility programs and DLLs are located in this directory. Proceed as follows:

- W – Select Start / Settings / Control Panel / System and click on the environment tab.
- W – Enter the path listed above to the PATH variable and click on the "Set" button.

W 2. Create one or more source files with control statements using an ASCII editor such as the NOTEPAD for the openUTM generation. You must observe the information stated in [section "Order of the control statements" on page 257](#) and in [section "Format of the control statements" on page 258](#).

W 3. Create the *filebase* directory (project directory) in which openUTM stores the KDCFILE and other application-specific files. You must create the directory **before** starting KDCDEF. *filebase* is the directory that you specified in the MAX statement in the FILEBASE= operand.

W 4. Now start the KDCDEF tool. Open a command mode window to do this with Start / Programs / Command Prompt. KDCDEF reads the KDCDEF control statements from *stdin* by default, i.e. directly from the command prompt. Enter the following to have KDCDEF read the control statements from a file (e.g. *definput.txt*):

W `kdcdef < definput.txt`

W or start KDCDEF with `kdcdef` and pass the file using the KDCDEF statement OPTION:

W `OPTION DATA=definput.txt`

W The messages and logs from KDCDEF are written to *stdout* and *stderr*, i.e. everything is displayed on the screen if you have not redirected the output. You can redirect the output to a file as follows (you can select any name you want for the files):

W `kdcdef < definput.txt 2>def.err 1>def.prot`

W All UTM messages are recorded in *def.err* and *def.prot* contains the complete log of the KDCDEF run.

6.2.2 Order of the control statements

Apart from the following exceptions, the control statements can be entered in any order. Apart from END and UTMD, all control statements can be entered several times.

- The END statement is always specified last, and concludes the sequence of control statements.
- In the OPTION statement, the last parameter value specified always applies.
- The order of the AREA statements indicates the order in which these areas must be specified in the parameter list and declared in the program unit (e.g. in the LINKAGE-SECTION in COBOL). See the openUTM manual “Programming Applications with KDCS”.
- The sequence of the EXIT statements with USAGE=START and USAGE=SHUT defines the sequence in which the programs of the event exits START and SHUT are executed when the application is started or shut down.
- The master LTERM of a LTERM bundle must be generated before the slave LTERMs of this LTERM bundle.
- The primary LTERM of a LTERM group must be generated before the alias LTERMs of this LTERM group.
- B** ● The DEFAULT statement refers only to the control statements entered thereafter.
- B**
B
B ● Load modules are loaded in the same order as that in which the LOAD-MODULE statements are entered. Refer to the LOAD-MODULE statement on [page 339](#) and openUTM manual “Using openUTM Applications under BS2000/OSD”.
- X/W**
X/W ● Shared objects/DLLs are loaded in the same order as that in which the SHARED-OBJECT statements are entered.

6.2.3 Format of the control statements

All KDCDEF control statements (apart from the DEFAULT statement under BS2000/OSD) have the following format:

control-statement *_operand1, operand2,...*

- *control-statement* can be entered starting in column 1 or later.
- *control-statement* must be separated from the operands by at least one blank.
- Each line of the control statement can be up to 240 characters in length. The control statements can be up to 3096 characters in length when continuation lines are used (see [section “Continuation lines in control statements” on page 258](#)).
- Comments can be inserted using the statement REMARK or by entering an asterisk (*) in column 1.
- The EJECT statement initiates a page feed in the log. The EJECT line itself is not logged.

6.2.4 Continuation lines in control statements

A control statement for the KDCDEF generation tool can consist of one or more lines, in which the hyphen (-) or backslash (\) can be used as the continuation character. In other words, if the last character of a line (apart from blanks) is a hyphen or a backslash, KDCDEF interprets the following line as belonging to the last statement specified. The continuation line can be entered starting in column 1 or later.

Each control statement can be up to 3096 characters in length, excluding comment lines, continuation characters, and blanks after the continuation character.

All comment lines must be marked with REMARK or an * in column 1.

6.2.5 Syntax and plausibility checks

KDCDEF carries out syntax and plausibility checks for all control statements entered. If KDCDEF does not detect any serious errors, then KDCDEF creates the KDCFILE and/or the source code for the ROOT tables, depending on what you have specified in OPTION.

In the case of UTM cluster applications, the UTM cluster files are also created where necessary.

KDCDEF always executes the plausibility checks for all control statements. If only one ROOT table source is created in a KDCDEF run, for example, then KDCDEF also checks the control statements that only affect the KDCFILE.

For this reason you should execute every KDCDEF run using all generation information, regardless of whether on the source code for the ROOT tables or only the KDCFILE is to be created.

Inconsistencies arising during the creation of the ROOT table module and KDCFILE that would otherwise only be detected once the application is started can be detected much earlier when complete plausibility checks are used. Errors are avoided.

6.2.6 KDCDEF logging

To improve legibility, KDCDEF logging can be structured as follows:

- Comments can be inserted in the KDCDEF log:
 - KDCDEF control statement "*comment*"
The comment entered after a KDCDEF control statement must not contain quotes.
 - * *comment* or **REMARK** *comment*
A * or REMARK in column 1 creates a comment line with a line number.
- Markers can be inserted in front of KDCDEF control statements, and must be preceded by a period (*.marker*). *marker* can be up to eight alphanumeric characters in length, and must begin with a letter.
- The EJECT statement initiates a page feed in the log. The EJECT line itself is not logged.

6.2.7 Format and uniqueness of object names

When configuring objects of the application, you must assign names to the objects. These names are then used by openUTM or the user to address specific objects. The following conditions should be borne in mind when assigning names:

- You must not use a reserved name.
- The object name must be unique within that particular object class.
- The name must not exceed the defined maximum length, and must contain permitted characters only (format).

6.2.7.1 Reserved names

Please note the comments below in order to ensure that the allocation of reserved names does not result in unexpected, undefined UTM application behavior:

- Names which start with KDC are reserved for the transaction codes of the event services, the administration commands (KDCADM), the Dead Letter Queue and the SAT administration (BS2000/OSD) and should only be used for such objects.

B
B

This does not apply to the load modules belonging to a UTM application under BS2000/OSD.

B
B

- Under BS2000/OSD program unit names must not start with prefixes which are used for runtime systems such as IT, IC etc.

X/W
X/W
X/W

- Under Unix systems and Windows systems the names of UTM objects must not start with KDC, KC, x, ITS or mF. External names (e.g. program unit names) must not start with 't_', 'a_', 'o_' or 's_' which are reserved for CMX (t_) or OSS (a_, o_, s_).

6.2.7.2 Format of names

The following conventions must be observed for names entered in KDCDEF control statements:

- The base name of the KDCFILE (MAX ...,KDCFILE=) must comply with the rules for file names of the operating system, under which the application is to run (for further information, see MAX statement on [page 368](#)).

- The names of LTERM partners, clients and printers, transaction codes and TAC queues etc. can be up to eight characters in length, where the following characters are permitted:

- A,B,C,...,Z
- 0,1,...,9
- #, @, \$

X/W

Under Unix systems and Windows systems, names may also contain lowercase letters (a,b,c,...,z). The names are case sensitive.

X/W

- Program names specified as entry/object names in the PROGRAM statement may be up to 32 characters long. This also applies for program names in TAC PROGRAM= and EXIT PROGRAM=.

The following characters are permitted in program names:

- A,B,C,...,Z
- 0,1,...,9
- #, @, \$

If other special characters are used, the program name must be enclosed in quotes, see below.

- Additional special characters in names:
 - Names of programs or passwords may also include other special characters such as "_" (underscore) and "-" (hyphen) if permitted by the particular system environment.
 - Load module names in BS2000/OSD (LOAD-MODULE) may also include the "." (period) and "-" (hyphen) characters.
 - Names that include other special characters (program names, passwords, etc.) must be enclosed in quotes.
Refer also, for example, to USER...,PROTECT-PW= on [page 538](#).

B

B

- Exceptions to name length rules:
 - Presentation and session selectors in the ACCESS-POINT and OSI-CON statements can be up to 16 characters in length.
 - program names can be up to 32 characters long.
 - Under BS2000/OSD, load module names in BLS generation can be up to 32 characters long; the names of common memory pools (Mpools) can be up to 50 characters long.

B

B

B

6.2.7.3 Number of names

A name is created for each of the following control statements:

ACCESS-POINT	MUX (BS2000/OSD)
BCAMAPPL	OSI-CON
CON	OSI-LPAP
EDIT	PROGRAM
KSET	PTERM
LOAD-MODULE (BS2000/OSD)	SHARED-OBJECT (Unix systems, Windows systems)
LPAP	TAC
LSES	TLS
LTAC	TPOOL
LTERM	USER
MASTER-OSI_LPAP	ULS
MASTER-LU61-LPAP	

Additional names are generated for the CLUSTER, CLUSTER-NODE; LTERM, MUX and TPOOL statements:

- 1 BCAMAPPL is also generated for the CLUSTER statement.
- 1 PTERM, 1 LTERM and 1 USER are also generated for each CLUSTER-NODE statement.
- If the application is generated without USER, two names are created for each LTERM statement.
- Two names are created for an LTERM statement belonging to a PTERM statement with PTYPE=APPLI, SOCKET, UPIC-R or UPIC-L if the implicit (connection) user belonging to this LTERM is not generated with an explicit USER statement.
- For each TPOOL statement, the number of names created is double that specified in the NUMBER= operand of the TPOOL statement. In the case of a TPOOL statement with PTYPE=APPLI, SOCKET, UPIC-R, the number of names created is **triple** that specified in NUMBER=.
- Two names are created for each MUX statement.

Furthermore, up to six additional names are created during generation, which are required by openUTM for event services (KDCSGNTC, KDCBADTC, KDCMSGTC, KDCMSGUS, KDCMSGLT, KDCAPLKS). The first three names can also be specified in a TAC statement. The last three names may not be specified.

If XATMI program units are generated for a UTM application, i.e. if API=(XOPEN,XATMI) is set in at least one TAC statement, then a TAC entry named KDCTXCOM and a PROGRAM entry named KDCTXRLB are created by openUTM.

The name KDCDLETQ is created for the dead letter queue during generation. The properties of this TAC queue can also be defined in a separate TAC statement.

Maximum values for names

The table below shows the maximum number of names that can be created using KDCDEF control statements. If this number is exceeded, then the generation is terminated.

	Group of KDCDEF control statements	Maximum number of generated names
	#USER + #APPLI + #LSES + #OSI-ACTIVE-ASSOCIATIONS + (2 * #TASKS) + 1	≤ 500000
B	#PTERM + #CON + TPOOLNR + #OSI-ASSOCIATIONS + #MUX ¹	≤ 500000
B	#LTERM + #LPAP + TPOOLNR + #OSI-LPAP + #TASKS + #MUX ¹ + 1	≤ 500000
X/W	#PTERM + #CON + TPOOLNR + #OSI-ASSOCIATIONS	≤ 500 000
X/W	#LTERM + #LPAP + TPOOLNR + #OSI-LPAP + #TASKS + 1	≤ 500 000
	#PROGRAM	≤ 32000
	#TAC + 4	≤ 32000
	#LSES	≤ 65000
	#CON	≤ 65000
	#KSET + 1	≤ 32000
	#LTAC	≤ 32000
	#MUX ¹	≤ 9999
	Total of all names + 2	≤ 32767

¹ Only supported under BS2000/OSD

Description of placeholders:

#statement Number of names generated using this KDCDEF statement

#APPLI Number of PTERM statements plus the TPOOLNR values of the TPOOL statements with PTYPE=APPLI/SOCKET/UPIC-R and UPIC-L (UPIC-L only under Unix systems and Windows systems)

In the case of UTM cluster applications, the values of #PTERM, #LTERM and #APPLI are each increased by the number of specified CLUSTER-NODE statements.

B #MUX Total number of generated MUX statements (only under BS2000/OSD)

#OSI-ACTIVE-ASSOCIATIONS

Number of active parallel OSI connections of the generated operand values (OSI-CON ...,ACTIVE=YES and associated OSI-LPAP ...,ASSOCIATIONS=*number*). This is the sum of all ASSOCIATIONS values in all OSI-LPAP statements.

#OSI-ASSOCIATIONS

#OSI-ACTIVE-ASSOCIATIONS plus the number of inactive parallel OSI-connections. (OSI-CON ...,ACTIVE=YES/NO and associated OSI-LPAP ...,ASSOCIATIONS=*number*). This is the sum of all ASSOCIATIONS values in all OSI-LPAP statements, including the values of OSI-LPAP statements for which backup connections are generated.

TPOOLNR Sum of all NUMBER= operands (number of LTERM partners in each LTERM pool) in all generated TPOOL statements

The following must also be noted:

- The number of names for #PROGRAM, #TAC, #LTERM, #PTERM, #USER, #KSET and #LTAC includes names generated statically and reserved names for objects that can be entered dynamically.
- The names of MASTER-LU61-LPAP statements must also be counted with #LPAP.
- The names of MASTER OSI-LPAP statements must also be counted for #OSI-LPAP.
- If the application was generated without USER statements, #USER must be replaced by #LTERM + TPOOLNR in the first condition.
- You can generate up to 100 ULS blocks and 100 TLS blocks.
- The number of generated user IDs (#USER) plus the number of entries intended for service stacking (defined in MAX NRCONV=) is restricted to a maximum of 500000.
- The number of generated user IDs (#USER) plus the number of entries intended for service stacking (MAX NRCONV) plus the maximum number of possible parallel asynchronous services (defined in MAX ASYNTASKS = (...*service_number*)) plus the number of entries reserved for sign on services (SIGNON CONCURRENT-TERMINAL-SIGNON) is restricted to a maximum of 665000.

6.2.7.4 Uniqueness of names and addresses

The objects of a UTM application are combined in shared name spaces which are defined for specific object types. The names and address of objects of the permitted types must be unique throughout the name class. A name or address must only be assigned once within the name class. There are three name classes:

Name class 1

- LTERM partners (statement LTERM *ltermname*)
- LTERM partners created by openUTM for the LTERM pools (statement TPOOL ..., LTERM=*ltermprefix*, NUMBER=*number*)
- transaction codes and TAC queues (statements TAC *tacname*)
- LPAP or OSI-LPAP partners for server-to-server communication (statements OSI-LPAP *osi_lpap_name* and LPAP *lpapname*)

Name class 2

- user IDs (statement USER *username*)
- sessions for distributed processing based on LU6.1 (statement LSES *sessionname*)
- connections and associations for distributed processing based on OSI TP (statement OSI-LPAP ..., ASSOCIATION-NAMES=, ASSOCIATIONS=)

Name class 3

- clients and printers (PTERM statement)
Clients are terminals, UPIC-clients, transport system applications (DCAM, PDN, CMX and socket applications), and UTM partner applications that do not use a higher-level protocol (LU6.1, OSI TP) during communication.
- name of the partner application for distributed processing based on LU6.1 (CON statement)
- name of the partner application for distributed processing based on OSI TP (OSI-CON statement)
- B** ● multiplex connections of a UTM application under BS2000/OSD (MUX statement)

The objects listed above are communication partners of the UTM application. openUTM must be able to uniquely identify these objects and the connections to them. For this purpose, it assigns a name triplet to each communication partner. This name triplet must be unique within the UTM application and consists of the following components:

- the name of the communication partner.
This is specified in *ptermname* in the PTERM statement, in *remote_appliname* in the CON statement, in TRANSPORT-SELECTOR= in the OSI-CON statement and in *name* in the MUX statement.

B

Under BS2000/OSD the BCAM name of the communication partner must be specified.

- the name of the system on which the communication partner is located.
This is specified in the PRONAM= operand of the PTERM, CON and MUX statements and in the NETWORK-SELECTOR= operand of the OSI-CON statement.
- the name of the local application via which the connection to the communication partner is established. This is specified in the BCAMAPPL= operand of the PTERM, MUX and CON statements and in the LOCAL-ACCESS-POINT= operand of the OSI-CON statement.

6.2.8 Result of the KDCDEF run

Depending on the entries made during generation, the KDCDEF generation tool creates the following:

- B ● *BS2000/OSD*:
 - B – the KDCFILE with the main file *filebase.KDCA* and, if dual-file operation is used, the duplicate file *filebase.KDCB*
 - B – the ROOT table source
 - B – the page pool *filebase.PnnA*, possibly with the duplicate *filebase.PnnB*
 - B – the restart area *filebase.RnnA*, possibly with the duplicate *filebase.RnnB*
- X/W ● *Unix systems and Windows systems*:
 - X/W – the main file KDCA in the *filebase* directory and, if dual-file operation is used, the duplicate file KDCB, also in the *filebase* directory
 - X/W – the ROOT table source in the form of a C/C++ source
 - X/W – the page pool *PnnA*, possibly with the duplicate *PnnB*, in the *filebase* directory
 - X/W – the restart area *RnnA*, possibly with the duplicate *RnnB*, in the *filebase* directory
- Additionally, if a UTM cluster application is being generated, see also the [chapter “Notes on generating a UTM cluster application” on page 63](#):
 - the cluster configuration file
 - the cluster user file
 - the cluster page pool files (a control file and one or more files for the user data)
 - the cluster GSSB file
 - the cluster ULS file

The format of the KDCFILE is described in detail in [section “The KDCFILE” on page 45](#).

KDCDEF outputs a message to SYSOUT (BS2000/OSD) or *stderr* (Unix systems, Windows systems) indicating whether the KDCFILE was created successfully and specifying the size of the KAA (KDC Application Area) occupied by the application. It also outputs a log containing the control statements and any error messages to SYSLST (BS2000/OSD) or *stdout* (Unix systems, Windows systems).

B Note for KDCDEF under BS2000/OSD

- B If the KDCDEF generation tool terminates abnormally due to an error, it sets process switch 3 (as occurs with all UTM tools). In this case, no files are generated apart from those created by the CREATE-CONTROL-STATEMENTS statement.

6.3 Inverse KDCDEF

The inverse KDCDEF function provided by openUTM is used to ensure that all changes made to the configuration dynamically during runtime are not lost when your application is regenerated. It creates control statements for the KDCDEF generation tool from the configuration data in the current KDCFILE.

Inverse KDCDEF generates control statements for object types that can be entered and deleted dynamically:

- **USER statements**

For all user IDs currently defined in the application. Inverse KDCDEF does not create USER statements for user IDs defined internally by UTM for the LTERM partners of clients of type UPIC-R, APPLI and SOCKET.

- **LTERM statements**

For all LTERM partners of the application which do not belong to an LTERM pool or to a multiplex connection (BS2000/OSD).

- **PTERM statements**

For all clients and printers entered in the configuration. No PTERM statements are created for clients that connect via an LTERM pool to the application or that belong to a multiplex connection.

- **PROGRAM statements**

For all program units and conversation exits currently defined in the application configuration.

- **TAC statements**

For all transaction codes and TAC queues of the application.

- **KSET statements**

For all key sets of the application.

- **CON statements**

For all LU6.1 connections of the application.

- **LSES statements**

For all LU6.1 session names of the application.

- **LTAC statements**

For all local transaction codes for VTV partner applications.

Control statements are also generated for objects of the types listed above, which were created statically in a previous KDCDEF generation. All modifications entered dynamically for these objects during runtime are taken into consideration.

Inverse KDCDEF does **not** create control statements for object types other than those listed above. Nor does it generate control statements for other components of the application or for application parameters.

It does **not** create control statements for objects that were dynamically deleted from the application configuration. After regeneration, these objects are thus permanently removed from the configuration. They do not occupy a table location and their object names are no longer reserved.

After regeneration with KDCDEF, the update tool KDCUPD does not transfer any application data from the old KDCFILE to the new KDCFILE, which relates to objects deleted dynamically. This applies even if the new KDCDEF generation includes an object with the same name and type as a deleted object. In particular, KDCUPD does not transfer any asynchronous jobs created by LTERM partners or user IDs that have since been deleted.

The USER statements created by inverse KDCDEF do not include any passwords. For user IDs generated with a password, inverse KDCDEF creates USER statements with the following format:

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

Once the KDCDEF run is complete and the new KDCFILE has been created, you must transfer the passwords of the user IDs to the new KDCFILE using the KDCUPD tool. This is also possible in a UTM-F application. For further information, see [chapter “The tool KDCUPD – updating the KDCFILE” on page 577](#).

It is not generally necessary to transfer the passwords with KDCUPD with UTM cluster applications. In UTM cluster applications, the current passwords are stored in the cluster user file and not in the KDCFILE.

You only need to transfer the passwords with KDCUPD if a new cluster user file has been generated and you wish to retain the passwords from the last application run.



In order to ensure that the KDCFILE contains the current passwords, the current information on all users must be read once (e.g. using WinAdmin) before the application is terminated.

6.3.1 Starting inverse KDCDEF

Inverse KDCDEF can be started online or offline.

The inverse KDCDEF run is started **online** by issuing the `KC_CREATE_STATEMENTS` call via the program interface for administration. Further information can be found in the openUTM manual “Administering Applications”. Inverse KDCDEF can only be started **offline** if the application is not running, i.e. outside the application runtime. Since inverse KDCDEF reads data from the `KDCFILE`, you must ensure that this data is not modified during the inverse KDCDEF run.



Inverse KDCDEF can be started offline by calling the KDCDEF generation tool and issuing the control statement **CREATE-CONTROL-STATEMENTS**. This statement is described on [page 318](#).

You can start inverse KDCDEF such that KDCDEF control statements are created either for all permitted object types, or only for those object types combined in the object groups `CON`, `DEVICE`, `KSET`, `LSES`, `LTAC`, `PROGRAM` and `USER`.

- `CREATE-CONTROL-STATEMENTS *ALL`
KDCDEF control statements are created for all objects of type `TAC`, `PROGRAM`, `PTERM`, `LTERM` `USER`, `KSET`, `LTAC`, `CON` and `LSES`.
- `CREATE-CONTROL-STATEMENTS DEVICE`
`LTERM` and `PTERM` statements are created for `LTERM` partners, clients and printers.
- `CREATE-CONTROL-STATEMENTS PROGRAM`
`PROGRAM` and `TAC` statements are created for program units, conversation exits, and transaction codes.
- `CREATE-CONTROL-STATEMENTS USER`
`USER` statements are created for user IDs.
- `CREATE-CONTROL-STATEMENTS KSET`
`KSET` statements are created for key sets.
- `CREATE-CONTROL-STATEMENTS LTAC`
`LTAC` statements are created for transaction codes. These are used to start the service programs in partner applications.
- `CREATE-CONTROL-STATEMENTS CON`
`CON` statements are created for transport connections to remote LU6.1 applications.

- CREATE-CONTROL-STATEMENTS LSES

LSES statements are created for assigning new LU6.1 session names.

6.3.2 Result of inverse KDCDEF

With inverse KDCDEF, you can define whether all control statements are to be written to the same file, or whether the control statements of a particular object group are to be written to a separate file. When starting inverse KDCDEF, you specify the name(s) of the file(s) to be created. Provided a file with this name does not exist, the file is created automatically. If a file with this name already exists, you can define whether this file is to be overwritten or updated.

The CREATE-CONTROL-STATEMENTS statement is applied immediately. You can therefore issue the OPTION statement immediately after the CREATE-CONTROL-STATEMENTS statement in the same KDCDEF run. This transfers the files created by inverse KDCDEF to KDCDEF. For example:

```
:  
CREATE-CONTROL-STATEMENTS *ALL, TO-FILE=control_statements_file  
                                , MODE=CREATE, FROM-FILE=kdcfile  
OPTION DATA=control_statements_file  
:  
:  
END
```

The diagram below illustrates how you can transfer the files generated by inverse KDCDEF directly as input files to KDCDEF. However, you can also edit them, i.e. modify them before the KDCDEF run and pass them to KDCDEF later as part of a regeneration. In this case, you simply terminate the generated control statements with the END statement. You assign each generated input file to KDCDEF with the control statement OPTION DATA=*control_statements_file* before the start.

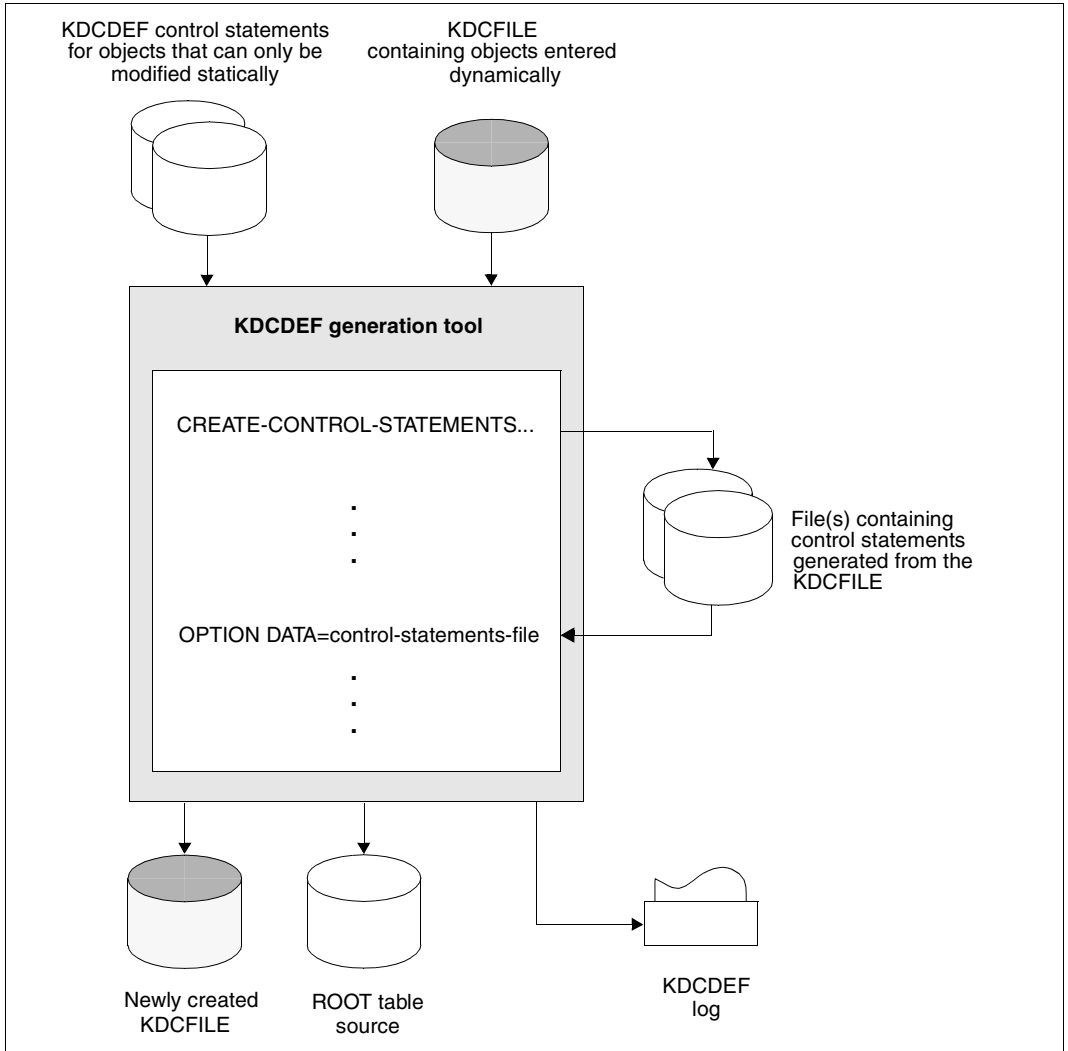


Figure 18: KDCDEF run with inverse KDCDEF

6.3.3 Creating KDCDEF control statements in upgrades

To enable inverse KDCDEF to read information from the KDCFILE, you must ensure that the KDCFILE was created with the same UTM version as the KDCDEF generation tool used for the inverse KDCDEF run.

If you upgrade to a new openUTM version, the KDCDEF control statements must first be created in the previous version, i.e. you must start the inverse KDCDEF of the previous version. The generated files can then be used as input files for the KDCDEF of the new openUTM version.

6.4 Recommendations when regenerating an application

During the operation of a UTM application, it may become necessary to regenerate the application.

In the case of UTM cluster applications, there are changes that can be made with a new generation of the KDCFILE with a running UTM cluster application and changes that can only be made when the UTM cluster application has been completely terminated.



A list of changes that require the UTM cluster application to be completely terminated before the application is started with the new KDCFILE can be found in the following manuals:

- openUTM manual “Using openUTM Applications under BS2000/OSD”
- openUTM manual “Using openUTM Applications under Unix systems and Windows systems”

Possible reasons for initiating a new KDCDEF run are listed below:

- to adjust the maximum values defined during generation
- to create new objects for distributed processing based on LU6.1 or OSI TP, because the server group is to be expanded during distributed processing
A KDCDEF run is only needed for distributed processing based on LU6.1 if it is necessary to insert LPAP objects. Objects of the types CON, LSES and LTAC, on the other hand, can be created with dynamic administration (provided that sufficient table entries were reserved with the RESERVE statement).
- to enter new load modules (BS2000/OSD), shared objects (Unix systems) or DLLs (Windows systems) in the application program
- in cases where table locations reserved for the dynamic entry of objects in the configuration are occupied, to extend the table or to remove objects marked for deletion in order to release the table locations and object names for further use

The application downtime associated with regeneration can be reduced by observing the following recommendations:

- When generating your application for the first time, split the KDCDEF control statements between various files depending on whether the objects involved can only be generated statically or can be entered dynamically. These files can then be provided to KDCDEF as input files using the `OPTION DATA=` statement.
- The control statements `USER`, `LTERM`, `PTERM`, `PROGRAM TAC`, `CON`, `KSET`, `LSES` and `LTAC` should be entered separately in files in accordance with the various object groups. When regenerating the application, you can simply replace these files with those created by inverse KDCDEF (`DEVICE`, `PROGRAM`, and `USER`, `CON`, `KSET`, `LSES` and `LTAC`). Further information can be found in the description of the `CREATE-CONTROL-STATEMENTS` statement on [page 318](#).

Before regenerating the application or initiating the inverse KDCDEF run, it is recommended that you dynamically delete all objects that are to be excluded from the new configuration (`KC_DELETE_OBJECT` call). Further information can be found in the openUTM manual “Administering Applications”.



In UTM cluster applications, objects that can be administered dynamically must always be deleted using the administration facilities. Only deleting the objects in the KDCDEF source leads to inconsistencies in the individual node applications of the UTM cluster application.

Compared to the manual deletion of control statements from the input file for the KDCDEF run, dynamic deletion offers the following advantages:

- If an object is manually deleted from the input file during regeneration, and another object is defined with the same name and type but with different properties in the same generation run, the KDCUPD tool does not recognize these as two different objects, and transfers the data of the deleted object to the `KDCFILE`. This can be avoided by dynamically deleting the object beforehand, and then creating an object with the same name and type during regeneration. In this case, KDCUPD will recognize these as two different objects, and will not transfer the data of the old object into the new `KDCFILE`.
- The manual deletion of KDCDEF statements from the KDCDEF input file is both tedious and prone to errors. During deletion, you must look out for dependencies between objects and thus between the KDCDEF statements. If dependencies are inadvertently overlooked, the KDCDEF run will have to be repeated thus increasing downtimes.
- The processes performed during regeneration can be automated. Within a single procedure, you can call inverse KDCDEF, transfer the generated files directly to KDCDEF, and call the KDCUPD update tool. This fully automatic procedure minimizes downtimes during regeneration.

To prevent undesirable repercussions from dynamic deletion, make sure for instance that there are no jobs pending for objects deleted or loaded dynamically during runtime.

6.5 KDCDEF control statements

ABSTRACT-SYNTAX - define the abstract syntax

The ABSTRACT-SYNTAX control statement is only required if you want to define your own Application Context for communication via the OSI-TP protocol (see the APPLICATION-CONTEXT statement on [page 285](#)).

ABSTRACT-SYNTAX defines a local name for an abstract syntax, and to assign an object identifier and the transfer syntax selected for transferring the user data. Since openUTM automatically generates the abstract syntaxes CCR, UDT, XATMI and UTMSEC. Therefore, they need not be explicitly generated using the ABSTRACT-SYNTAX statement. It is possible to generate up to 50 abstract syntaxes, including those generated implicitly by openUTM.

```
ABSTRACT-SYNTAX_ abstract_syntax_name
                ,OBJECT-IDENTIFIER=object_identifier
                [ ,TRANSFER-SYNTAX=transfer_syntax_name ]
```

abstract_syntax_name

Local name for an abstract syntax up to eight characters in length. This name must be unique within the UTM application.

abstract_syntax_name must be specified in MGET/MPUT or FGET/FPUT when sending or receiving data in this abstract syntax.

OBJECT-IDENTIFIER=object_identifier

Object identifier of the abstract syntax specified as follows:

object_identifier=(*number1,number2, ... ,number10*)

number is a positive integer in the range 0 to 67108863. For *object_identifier*, you can specify two to ten integers enclosed in parentheses, each of which is separated by a comma. The number of integers entered and their positions are relevant. Instead of the integer itself, you can also specify the symbolic name assigned to this integer. The table on [page 97](#) shows the permitted values for *number* at the various positions.

object_identifier must be unique with the UTM application, i.e. another abstract syntax must not be generated with the same object identifier.

TRANSFER-SYNTAX=transfer_syntax_name

Name of a transfer syntax defined using the TRANSFER-SYNTAX control statement.

Default: BER (Basic Encoding Rules)

openUTM automatically generates the abstract syntaxes CCR, UDT, XATMI and UTMSEC, which are defined as follows:

Generation of “CCR”:

```
ABSTRACT-SYNTAX_CCR,           -
    OBJECT-IDENTIFIER=(2, 7, 2, 1, 2), -
    TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

(joint-iso-ccitt, ccr, abstract-syntax, apdus, version2)

Generation of “UDT”:

```
ABSTRACT-SYNTAX_UDT,           -
    OBJECT-IDENTIFIER=(1, 0, 10026, 6, 1, 1), -
    TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

(iso, standard, tp, udt, generic-abstract-syntax, version)

Generation of “XATMI”:

```
ABSTRACT-SYNTAX_XATMI,           -
    OBJECT-IDENTIFIER=(1, 2, 826, 0, 1050, 4, 1, 0), -
    TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

(iso, national-member-body, bsi, disc, xopen, xatmi, apdus-abstract-syntax, version1)

Generation of “UTMSEC”:

```
ABSTRACT-SYNTAX_UTMSEC,           -
    OBJECT-IDENTIFIER=(1, 3, 0012, 2, 1107, 1, 6, 1, 2, 0), -
    TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

(iso, identified-organisation, icd-ecma, member-company, siemens-units, sni, transaction-processing, utm-security, abstract-syntax, version)

ACCESS-POINT - create an OSI TP access point

The ACCESS-POINT control statement is required only for communication based on the OSI TP protocol. It defines a local access point to the services of OSI TP.



If you issue more than one ACCESS-POINT statement per application, then KDCDEF outputs warning K492.

Using the information specified in the ACCESS-POINT statement, a partner application can address the local application.

You specify the following parameters for an access point in the ACCESS-POINT statement:

- Address of the access point within the local system
The address of the access points consists of the presentation selector, session selector and transport selector components.

The address specifications must be coordinated with the communication partners. The TRANSPORT-SELECTOR specification is mandatory in all cases.

X/W
X/W
X/W
X/W
X/W

Unix systems and Windows systems:

On Unix systems and Windows systems the address of the access point also comes from the LISTENER-PORT, T-PROT, and TSEL-FORMAT components.

See [section “Providing address information for the CMX transport system \(Unix systems and Windows systems\)” on page 110](#) for more information.

- Application Entity Qualifier
You can define an application entity qualifier (AEQ) as additional address information. The application entity qualifier (AEQ) is combined with the application process title (APT) defined in the UTMD statement to form the application entity title (AET). The AET is a globally unique name for an application entity within the OSI TP environment. During transaction-oriented processing, the partner application requires the AET of the local UTM application in order to establish a connection. Similarly, the local application requires the AET of the partner application. It must be specified in the OSI-LPAP control statement that defines the partner application. The transport selector for the access point is still a mandatory entry.

X/W
X/W
X/W
X/W
X/W

- Listener ID (Unix systems, Windows systems)
Under Unix systems and Windows systems the access point is assigned a listener ID if the application accesses the network using multi-threaded processes. Network connections managed by the same network process can thus be combined. The connection to the network is always set up using multi-threaded processes under Windows systems.

Each ACCESS-POINT is signed on to the transport system when the application is started (provided this is possible), and is not signed off until the application is terminated.

```

ACCESS-POINT_ access_point_name
    [ ,APPLICATION-ENTITY-QUALIFIER=aequalifier ]
    ,PRESENTATION-SELECTOR={ *NONE |
                            (C'c' [ ,STD | EBCDIC | ASCII ]) |
                            X'x' }
    ,SESSION-SELECTOR={ *NONE |
                       (C'c' [ ,STD | EBCDIC | ASCII ]) |
                       X'x' }
    ,TRANSPORT-SELECTOR=C'c'

```

X/W

further operands for Unix systems and Windows systems

X/W

[,LISTENER-ID=number]

X/W

[,LISTENER-PORT=number]

X/W

[,T-PROT=(RFC1006)]

X/W

[,TSEL-FORMAT={ T | E | A }]

access_point_name

Name of the OSI TP access point, which is then used to identify the access point in the local UTM application.

access-point_name can be up to eight characters in length. *access-point_name* must be unique within the local UTM application.

APPLICATION-ENTITY-QUALIFIER=aequalifier

Address component of the application entity title (AET). The AET is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET in order to establish a connection.

An application entity qualifier (AEQ) can be specified only if an application process title (APT) is also defined for the application in the UTMD statement.

However, an APT need not necessarily be assigned an AEQ. If AEQ is **not** defined, the access point has no application entity title (AET), i.e. it cannot be used for transaction management (commit functional unit).



If the application context of an OSI-LPAP partner that operates via this access point (OSI-CON statement) contains the CCR syntax, you must enter an application entity qualifier here.

For *aequalifier*, specify a positive integer. *aequalifier* must be unique within the application, i.e. *aequalifier*=integer1 must not be specified as the AEQ in any other ACCESS-POINT statement.

Minimum value: 1

Maximum value: 67108 863 ($2^{26}-1$)

X/W	LISTENER-ID=number
X/W	This assigns a listener ID to the access point as administrative information used for multi-threaded network access.
X/W	
X/W	Listener IDs can be specified for access points and application names. Further information can be found in the description of the BCAMAPPL statement.
X/W	
X/W	Multi-threaded network accesses (MAX ...,NET-ACCESS=
X/W	MULTI-THREADED) allows you to manage several connections in a single network process. All connections with the same listener ID are managed within the same network process. Listener IDs also allow you to distribute the management of network access over a number of network processes.
X/W	
X/W	If you do not explicitly specify a listener ID, openUTM assigns the value 0 and combines all connections without a listener ID into a single network process.
X/W	
X/W	Default value: 0
X/W	Minimum value: 0
X/W	Maximum value: 32767
X/W	
X/W	BCAMAPPL names that were created for communication via the socket interface (native TCP/IP) use separate network processes. Their listener IDs comprise a separate number space. i.e. they are administered in a different network process even if they have the same listener ID as this access point.
X/W	
X/W	LISTENER-PORT=number
X/W	Port number of the access point for establishing TCP/IP connections.
X/W	
X/W	Permitted values: 1025 - 32767
X/W	
X/W	Default: 0 (i.e. no port number)
X/W	102 for OPTION CHECK-RFC1006
X/W	
X/W	If OPTION CHECK-RFC1006=YES, then a port number must be entered for LISTENER-PORT.

PRESENTATION-SELECTOR=

Presentation selector for the address of the OSI TP access point.

- *NONE The address of the OSI TP access point does not contain a presentation selector.
- C'c' The presentation selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The presentation selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted.
- STD The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix systems and Windows systems = ASCII).

Default: STD
- EBCDIC The characters are interpreted as EBCDIC code.
- ASCII The characters are interpreted as ASCII code.
- X'x' The presentation selector is entered in the form of a hexadecimal number (x). The value specified for *x* can be up to 32 hexadecimal digits (≅ 16 bytes) in length. You must enter an even number of hexadecimal digits.

SESSION-SELECTOR=

Session selector for the address of the OSI TP access point.

- *NONE The address of the OSI TP access point does not contain a session selector.
- C'c' The session selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The session selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted.
- STD The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix systems and Windows systems = ASCII).

Default: STD
- EBCDIC The characters are interpreted as EBCDIC code.
- ASCII The characters are interpreted as ASCII code.

X'x' The session selector is entered in the form of a hexadecimal number (x). The value specified for x can be up to 32 hexadecimal digits ($\hat{=}$ 16 bytes) in length. You must enter an even number of hexadecimal digits.

TRANSPORT-SELECTOR=C'c'

Transport component for the address of the OSI TP access point.

The specification of T-SEL=C'c' is mandatory.

You can enter up to eight printable characters. Permitted characters include uppercase letters, numbers, and the special characters \$, # and @.

Hyphens are not permitted. The first character of the name must be an uppercase letter.

The name defined in T-SEL must be unique in the local UTM application. It must not be the same name as the primary application name specified in MAX APPLNAME, a BCAMAPPL name or the name specified with a T selector in an ACCESS-POINT control statement.

B

BS2000/OSD:

B

T-SEL= specifies the local BCAM application name. The transport selector must be unique in the local system for each host.

B

X/W

Unix systems and Windows systems:

X/W

You must match T-SEL to the transport selector of the OSI TP partner. If, for example, the partner is a UTM application, the specification in T-SEL must match the transport selector of the OSI-CON statement on the partner.

X/W

X/W

X/W

T-PROT=

Address formats of the T-selectors of the access point

X/W

Further Information can be found in the "CMX User Guide" or in the PCMX online help system.

X/W

X/W

RFC1006

Address format RFC1006, ISO transport protocol based on TCP/IP and RFC1006 convergence protocol.

X/W

X/W

Default: RFC1006

X/W TSEL-FORMAT=
X/W Format indicator of the T-selectors of the access point
X/W (operand TRANSPORT-SELECTOR)

X/W The format indicator specifies the encoding of the T-selectors in the
X/W transport protocol. You will find more information in the "CMX User Guide"
X/W and in the PCMX online help system.

X/W T TRANSDATA format (encoded in EBCDIC)

X/W E EBCDIC character format

X/W A ASCII character format

X/W Default:
X/W T if the character set of the T-selector corresponds to the TRANSDATA
X/W format
X/W E in all other cases

X/W It is recommended to specify a value explicitly for TSEL-FORMAT.

ACCOUNT - define the accounting functions

The ACCOUNT control statement allows you to define:

- whether the accounting or calculation phase of the UTM accounting is to be activated at the start of the UTM application,
- when an accounting record is written,
- the weighting with which resources are to be evaluated in the accounting phase.

If the ACCOUNT control statement is not specified, then this has the same effect as ACCOUNT ACC=NO. Only the first ACCOUNT statement of a KDCDEF run is evaluated.

UTM accounting can also be activated and deactivated via the administration, even if no ACCOUNT statement is issued in the KDCDEF generation. In this case the default values apply.

You may only specify the ACCOUNT statement once within a KDCDEF run.



The UTM accounting functions and the format of accounting records written by openUTM are described in the openUTM manual “Using openUTM Applications”.

```
ACCOUNT_ ACC={ YES | NO | CALC }
          [ ,CPUUNIT=cpuunit ]
          [ ,IOUNIT=iounit ]
          [ ,MAXUNIT=maxunit ]
          [ ,OUTUNIT=outunit ]
```

ACC=	specifies which UTM accounting functions are to be executed. ACC is a mandatory operand.
YES	openUTM is to activate the accounting phase of UTM accounting after the application start.
NO	The accounting functions are not activated after the application start. You can switch on the accounting functions during live operation using the administration command KDCAPPL ..., ACC=ON or via the program interface for administration (see the openUTM manual “Administering Applications”).
CALC	openUTM is to activate the calculation phase after the application start.

CPUUNIT=cpuunit

specifies the weighting with which a CPU second is evaluated in the accounting phase of the UTM accounting. Fractions of a CPU second are billed proportionally.

You must enter an integer here.

Default value: 0

Minimum value: 0

Maximum value: 32767

IUNIT=iunit

specifies the weighting with which 100 disk I/Os are evaluated in the accounting phase.

Fractions of 100 inputs/outputs are billed accordingly.

You must enter an integer here.

Default value: 0

Minimum value: 0

Maximum value: 32767

X/W
X/W
X/W



Unix systems and Windows systems:

This operand is not used because these operating systems do not provide information on disk I/O.

MAXUNIT=maxunit

specifies the number of accounting units at which openUTM is to create an accounting record for a particular user (USER).

You must enter an integer here.

Default value:

99 999 999 (=10⁸-1)

i.e. an accounting record is normally created only on connection shutdown.

Minimum value: 1

Maximum value: 99 999 999 (=10⁸ - 1)

OUTUNIT=outunit

specifies the weighting with which a print job (FPUT NE) is evaluated for accounting purposes.

You must enter an integer here.

Default value: 0

Minimum value: 0

Maximum value: 4095

APPLICATION-CONTEXT - define the application context

The APPLICATION-CONTEXT control statement is required only for communication based on the OSI TP protocol. You only have to issue the APPLICATION-CONTEXT statement if you want to define an additional application context.

It allows you to define the application context used for communication via OSI TP. The application context determines the rules governing data transfer between the communication partners. It defines how the user data is encoded for transfer, and the format in which data is transferred. The application context must be coordinated with the partner.

The APPLICATION-CONTEXT statement enables you to define a local name for an application context, and to assign an object identifier and the abstract syntaxes belonging to this application context.

openUTM generates the standard application contexts UDTAC, UDTDISAC, XATMIAC, UDTCCR, UDTSEC and XATMICCR.

```
APPLICATION-CONTEXT_ application_context_name
                        ,OBJECT-IDENTIFIER=object_identifier
                        ,ABSTRACT-SYNTAX={ abstract_syntax_name |
                                           (abstract_syntax_name,...) }
```

application_context_name

A local name for an application context up to eight characters in length.

application_context_name must be unique within the UTM application.

OBJECT-IDENTIFIER=object_identifier

Object identifier of the application context specified as follows:

object_identifier=(*number1,number2, ... ,number10*)

number is a positive integer in the range 0 to 67108863. For *object_identifier*, you can specify two to ten integers enclosed in parentheses, each of which is separated by a comma. The number of integers entered and their positions are relevant.

Instead of the integer itself, you can also specify the symbolic name assigned to this integer. The table on [page 97](#) shows the permitted values for *number* at the various positions.

object_identifier must be unique within the UTM application, i.e. another application context must not be generated with the same object identifier.

ABSTRACT-SYNTAX=

Abstract syntax assigned to the application context for the transfer of user data.

abstract_syntax_name

Name of an abstract syntax defined using the ABSTRACT-SYNTAX control statement.

(abstract_syntax_name, ..., abstract_syntax_name)

List of up to nine abstract syntaxes separated by commas. Each abstract syntax specified in *abstract_syntax_name* must be defined beforehand using the ABSTRACT-SYNTAX statement.

The default UTM syntaxes CCR, UDT, XATMI, and UTMSEC need not be explicitly generated.

To work with transaction processing, a application context must be selected that contains the abstract syntax CCR.

If sign-on data is to be passed in a APRO call, then a application context must be selected that contains the abstract syntax UTMSEC.

If both partners use the XATMI interface, then a application context must be selected that contains the abstract syntax XATMI.

openUTM automatically generates the application contexts UDTAC, UDTDISAC, XATMIAC, UDTCCR, UDTSEC and XATMICCR, which are defined as follows:

Generation of "UDTAC":

```
APPLICATION-CONTEXT_UDTAC,           -
    OBJECT-IDENTIFIER=(1, 0, 10026, 6, 2),   -
    ABSTRACT-SYNTAX=UDT
```

Symbolic representation of the object identifier:

(iso, standard, tp, udt, application-context)

Generation of "UDTDISAC":

```
APPLICATION-CONTEXT_UDTDISAC,       -
    OBJECT-IDENTIFIER=(1, 0, 10026, 6, 2, 1),   -
    ABSTRACT-SYNTAX=UDT
```

Symbolic representation of the object identifier:

(iso, standard, tp, udt, application-context, with-tp)

Generation of “XATMIAC”:

```
APPLICATION-CONTEXT_XATMIAC, -
    OBJECT-IDENTIFIER=(1, 2, 826, 0, 1050, 4, 2, 1), -
    ABSTRACT-SYNTAX=(XATMI)
```

Symbolic representation of the object identifier:

```
(iso, national-member-body, bsi, disc, xopen, xatmi, application-context,
atp11-21-31)
```

Generation of “UDTCCR”:

```
APPLICATION-CONTEXT_UDTCCR, -
    OBJECT-IDENTIFIER=(1, 0, 10026, 6, 2), -
    ABSTRACT-SYNTAX=(UDT, CCR)
```

Symbolic representation of the object identifier:

```
(iso, standard, tp, udt, application-context)
```

Generation of “UDTSEC”:

```
APPLICATION-CONTEXT_UDTSEC, -
    OBJECT-IDENTIFIER=(1, 3, 0012, 2, 1107, 1, 6, 1, 3, 0), -
    ABSTRACT-SYNTAX=(UDT, UTMSEC, CCR)
```

Symbolic representation of the object identifier:

```
(iso, identified-organisation, icd-ecma, member-company, siemens-units,
sni, transaction-processing, utm-security, application-context, version)
```

Generation of “XATMICCR”:

```
APPLICATION-CONTEXT_XATMICCR, -
    OBJECT-IDENTIFIER=(1, 2, 826, 0, 1050, 4, 2, 1), -
    ABSTRACT-SYNTAX=(XATMI, CCR)
```

Symbolic representation of the object identifier:

```
(iso, national-member-body, bsi, disc, xopen, xatmi, application-context,
atp11-21-31)
```

AREA - define additional data areas

The AREA statement allows you to define the name, properties, and sequence of additional shareable data areas. The structure of these areas is not defined by openUTM and can be defined as chosen. The addresses of such areas are passed to the program unit as parameters at the start of the program with the address of the communication area and the standard primary working area.



You have an alternative to administering areas with openUTM using AREA statements in most programming languages (especially under COBOL and C/C++). The alternative is to declare areas as external data areas and to access these areas from the program units. This option offers a number of benefits compared with AREAs. You will find more information on this subject in the openUTM manual "Programming Applications with KDCS".

Each area to be defined in openUTM must be defined in a separate AREA statement. The order of the AREA statements indicates the order in which these areas must be specified in the parameter list and declared in the program unit (e.g. under BS2000/OSD in the LINKAGE-SECTION under COBOL). If the area defined on the n -th location is required, then all areas in the parameter list and in the data declaration must be specified or declared up to this area.

It is possible to specify up to 99 AREA statements in a single generation run irrespective of the operating system.



AREAs in cluster applications are local to the node, i.e. each node application has its own instance of each AREA.

B **Generating areas under BS2000/OSD****B** Under BS2000/OSD areas can be created:

- B** ● in the global common memory pool (for all applications).
- B** ● in the local common memory pool (for all application processes started under the same user ID).
- B** ● in non-privileged subsystems.
- B** ● in the linked application program.

B The following applies for the AREA statement:

- B** ● If you specify the operand LOAD-MODULE=, you must also write a LOAD-MODULE statement. Note that no load module may be referenced which has been generated with LOAD-MODULE ...LOAD-MODE=ONCALL.
- B** ● AREA statements that do not contain the LOAD-MODULE operand define data areas that are linked statically to the application program.
- B** ● The default values for AREA are set using the DEFAULT PROGRAM statement.

B AREA_L areaname
B { [,LOAD-MODULE=lmodname] |

B areaname Name of the area. *areaname* is an alphanumeric value up to 32 characters in length. *areaname* must be a module.

B LOAD-MODULE=lmodname
B *lmodname* can be up to 32 characters in length.

B LOAD-MODULE= identifies the name of the load module to which the module is linked. This load module must be defined using the LOAD-MODULE statement, and must not be generated with the operand LOAD-MODE=ONCALL.

X/W **Generating areas under Unix systems and Windows systems**

X/W An area must be explicitly defined, compiled, and linked to the program unit as external C/C++ data structures.

X/W In the AREA statement, you can define whether the area is transferred directly to the program unit, or is accessed indirectly by means of a pointer. When accessing indirectly, a pointer must be supplied with the address of the area before the first program unit is started. You can set the addresses before compiling or during the application run in the event exit START, for example.

X/W AREA_ areaname
X/W [,ACCESS={ DIRECT | INDIRECT }]

X/W areaname Name of the area. *areaname* is an alphanumeric value up to 32 characters in length. *areaname* must be a module.

X/W ACCESS= Mode of access to the additional data area

X/W DIRECT The area is defined directly as a C data structure.

X/W Default: DIRECT

X/W INDIRECT The area is defined as a pointer. The pointer *areaname* must be supplied with the address of the area. It is possible to first set the address during the application run, e.g. you can store the address of a shared memory area in the pointer in the START event exit.

BCAMAPPL - define additional application names

The BCAMAPPL statement allows you to assign additional application names to the UTM application for client/server communication and distributed processing via LU6.1. You must also assign every BCAMAPPL name an address within the local system (T-selectors or station names) so that the application is addressable from the communication partner.

The primary application name of the UTM application is specified in the APPLNAME operand of the MAX statement. Please note the following:

- B ● *BS2000/OSD:*
B You may issue the BCAMAPPL statement only for additional BCAM names of the appli-
B cation. You must not issue a BCAMAPPL statement for the primary application name.
- X/W ● *Unix systems and Windows systems:*
X/W You will also need to issue a BCAMAPPL statement for the primary application name if
X/W you also wish to establish connections via this name to partner applications or clients.

It is necessary to generate additional application names for your UTM application if:

- parallel connections via LU6.1 are to be defined to other applications (distributed processing). In this event, additional application names must be generated in at least one of the applications involved.
- communication with a partner is to be done via the socket interface (native TCP/IP). You will need a separate BCAMAPPL name (with T-PROT=SOCKET) for the communication via the socket interface. This name cannot be used for communication via other transport protocols.
- B ● you select the transport protocol (not NEA) for a partner of a UTM application under
B BS2000/OSD generated with PTYPE=APPLI, PTYPE=UPIC-R, or generated as a
B partner of a LU6.1 application.
- B ● you establish multiplex connections to a partner of a UTM application under
B BS2000/OSD.
- X/W ● you want to establish connections via the RFC1006 protocol. In this case you must
X/W define a separate BCAMAPPL name for the communication via RFC1006.

The BCAMAPPL statement can be issued several times. However, to ensure that resources are not unnecessarily occupied, you should only generate as many BCAMAPPL statements (i.e. application names) as are necessary.



All K messages that refer to the UTM application name contain the name defined in the MAX statement rather than the BCAMAPPL statement.

B BCAMAPPL statement under BS2000/OSD

```

B BCAMAPPL_ appliname
B           ,LISTENER-PORT=number only allowed and mandatory for T-PROT=SOCKET
B           [ ,SIGNON-TAC={ *NONE | tacname } ]
B           [ ,T-PROT={ NEA | ISO | RFC1006 | SOCKET } ]
    
```

B appliname Additional BCAM name of the UTM application. *appliname* can be up to eight characters in length. *appliname* must not be identical to the application name you specified in MAX ...,APPLINAME= or in ACCESS-POINT ..., TRANSPORT-SELECTOR=.

B In addition, the name must be different from the application name that you specified in the BCAMAPPL operand of the CLUSTER statement.

B *appliname* must be unique in the local system for each host.

B LISTENER-PORT=number Only permitted for T-PROT=SOCKET. In this case, it is mandatory to specify the LISTENER-PORT.


B LISTENER-PORT specifies the port number on which openUTM waits for external connection establishment requests.

B All port numbers are allowed.

B A port number may only be used **once** in the local system to listen for connections being established via the socket interface (TCP-IP-APPLI). It may be when starting openUTM that some port numbers are already reserved by the system or other TCP/IP applications, or that privileged port numbers may not be used. In this case, the start of the UTM application is aborted.

B SIGNON-TAC = Specifies whether a sign-on service is to be started for connections that are established using the application names *appliname* (=transport system access point). If a sign-on service is to be started, you must specify the name of the transaction code via which the sign-on service is to be started.

B *NONE For connections to the UTM application that are to be established using the application name *appliname*, no sign-on service is to be started, regardless of whether the TAC is generated with KDCSGNTC or not.

B	tacname	Name of the service TAC started via the sign-on service.
B B B		The transaction code <i>tacname</i> must be generated using a TAC statement. In the TAC statement, you must not modify the following default settings for the transaction code:
B B B B B B B B		<ul style="list-style-type: none"> – API = KDCS, – CALL = FIRST or BOTH, – ENCRYPTION-LEVEL = NONE, – PGWT = NO, – TACCLASS = 0, – TYPE = D, – no limitation on data access authorizations, i.e. the operands ACCESS-LIST and LOCK may not be specified
B B B B B		For UPIC partners, the sign-on service is only started if UPIC=YES is generated in the SIGNON statement. In the case of UPIC partners, the signon service is not started when the connection is established. Instead, it is started before a UPIC conversation is started (see also SIGNON statement, OMIT-UPIC-SIGNOFF= parameter on page 481). For LU6.1 partners, no sign-on service is started.
B B B		<i>tacname</i> may not be assigned to a program (PROGRAM operand of a TAC statement) that is located in a load module generated with LOAD-MODE=ONCALL.
B B B B		Default: <ul style="list-style-type: none"> – KDCSGNTC as far as it is generated in the application (KDCSGNTC = standard sign-on service; generated with a TAC statement) – otherwise *NONE
B B B B B		<p>CAUTION!</p> <p>Those communication partners that establish their connection to the UTM application via the primary application name (generated in MAX APPLINAME=) can only have a sign-on service that is generated using the transaction code KDCSGNTAC.</p>

B	T-PROT=	Transport protocols to be used on the connections to partner applications that are established through this application name.
B		
B	NEA	An NEA transport protocol is used.
B		Default: NEA
B	ISO	An ISO transport protocol is used.
B		Whether or not an ISO transport connection can be established to this application and which transport protocol will actually be used depends on the generation of the transport system. As parallel connections are allowed for ISO transport connections although they are not supported by openUTM, openUTM accepts the connection of the contention winner (CON) or of the partner with the alphabetically smaller name pair (<i>ptermname</i> , processor name, PTERM statement) in case of a contention.
B		
B	RFC1006	TCP/IP is used with the RFC1006 convergence protocol.
B		RFC1006 must be used for communication with openUTM under Unix systems or Windows systems.
B		RFC1006 has the same effect as T-PROT=ISO in BS2000/OSD.
B		
B	SOCKET	Native TCP/IP is to be used as the transport protocol, i.e. communication is to be handled via the socket interface.
B		If you specify T-PROT=SOCKET, then you must define a port number in the LISTENER-PORT operand.
B		You will find more information on SOCKET on page 153ff.
B		

X/W BCAMAPPL statement under Unix systems and Windows systems

X/W Under Unix systems and Windows systems the operands *appliname*, LISTENER-PORT (TCP/IP port number), T-PROT (transport protocols used) and TSEL-FORMAT (format identifier) are used to specify the address.

X/W If the application accesses the network using multi-threaded processes, then you should define a listener ID. This listener ID is used to assign the network connections to a network process based on the BCAMAPPL name.

X/W BCAMAPPL_ appliname
 X/W [,LISTENER-ID=number]
 X/W [,LISTENER-PORT=number]
 X/W [,SIGNON-TAC={ *NONE | tacname }]
 X/W [,T-PROT={ SOCKET | RFC1006 }]
 X/W [,TSEL-FORMAT={ T | E | A }]

X/W **appliname** Additional name of the UTM application. *appliname* can be up to eight characters in length. *appliname* must not be identical to the application name you specified in ACCESS-POINT ...,TRANSPORT-SELECTOR=.

X/W In addition, the name must be different from the application name that you specified in the BCAMAPPL operand of the CLUSTER statement.

X/W *appliname* must be unique throughout the network.

X/W KDCDEF creates a T-selectors from *appliname* for the transport system. The T-selectors is part of the transport address of the application that is used to address the application from partner applications when establishing a connection.

X/W Exception:

X/W *appliname* is only relevant internally in UTM for T-PROT=SOCKET, e.g. for the administration. The name only needs to be unique within the application.

X/W	LISTENER-ID=number
X/W	This assigns a listener ID to the application name as administrative information used for multi-threaded network access.
X/W	
X/W	Listener IDs can be specified for application names and access points. Further information can be found in the description of the ACCESS-POINT statement.
X/W	
X/W	Multi-threaded network access (MAX ...,NET-ACCESS=MULTI-THREADED) allows you to manage several connections of threads in a single network process. All connections with the same listener ID are managed within the same network process. Listener IDs also allow you to distribute the management of network access over a number of network processes.
X/W	
X/W	BCAMAPPL names with T-PROT=SOCKET (communication via the socket interface) comprise a separate set of numbers, i.e. no BCAMAPPL names that were created for communication via the socket interface are combined with BCAMAPPL names and access points for other transport protocols in one network process, even if the listener ID is the same.
X/W	
X/W	All BCAMAPPL names for the socket interface with the same listener ID are always combined in a network process, even if NET-ACCESS=SINGLE-THREADED was generated.
X/W	
X/W	If multi-threaded network connection is generated and you do not specify a listener ID, then openUTM assigns the value 0 as the listener ID. openUTM then combines all connections without a listener ID. All connections without a listener ID that are not established via the socket interface are combined into a single network process, and all connections without a listener ID that are established via the socket interface are combined into another single network process.
X/W	
X/W	Default value: 0
X/W	Minimum value: 0
X/W	Maximum value: 32767
X/W	LISTENER-PORT=number
X/W	Port number of the UTM application for establishing TCP/IP connections.
X/W	
X/W	Permitted values for <i>number</i> :
X/W	T-PROT=RFC1006: Port numbers 1025 through 32767 are permitted.
X/W	T-PROT=SOCKET: Port numbers 1 through 65535 are permitted.
X/W	
X/W	With T-PROT=RFC1006 and OPTION CHECK-RFC1006=YES and with T-PROT=SOCKET, a port number must be specified for LISTENER-PORT. In all other cases, the default value is 0 (no port number).

X/W
X/W

– A port number may only be used **once** per processor to listen for connections being established via the socket interface (SOCKET).

X/W
X/W
X/W

– If the default value is used (port number 0), the default port number assigned by CMX is used internally. This can result in conflicts if, for example, the port is used by different applications.

X/W
X/W
X/W
X/W
X/W

SIGNON-TAC =

Specifies whether a sign-on service is to be started for connections that are established using the application names *appliname* (=transport system access point). If a sign-on service is to be started, you must specify the name of the transaction code via which the sign-on service is to be started.

X/W
X/W
X/W
X/W

*NONE

For connections to the UTM application that are to be established using the application name *appliname*, no sign-on service is to be started, regardless of whether the TAC is generated with KDCSGNTC or not.

X/W

tacname

Name of the service TAC started via the sign-on service.

X/W
X/W
X/W

The transaction code *tacname* must be generated using a TAC statement. In the TAC statement, you must not modify the following default settings for the transaction code:


X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W

- API = KDCS,
- CALL = FIRST or BOTH,
- ENCRYPTION-LEVEL = NONE,
- PGWT = NO,
- TACCLASS = 0,
- TYPE = D,
- no limitation on data access authorizations, or in other words, the operands ACCESS-LIST and LOCK may not be specified

X/W
X/W
X/W
X/W
X/W
X/W

For UPIC partners, the sign-on service is only started if UPIC=YES is also generated in the SIGNON statement. In the case of UPIC partners, the signon service is not started when the connection is established. Instead, it is started before a UPIC conversation is started (see also SIGNON statement, OMIT-UPIC-SIGNOFF= parameter on [page 481](#)).

For LU6.1 or OSI TP partners, no sign-on service is started.

X/W		Default:
X/W		– KDCSGNTC as far as it is generated in the application (KDCSGNTC =
X/W		standard sign-on service; generated with a TAC statement)
X/W		– otherwise *NONE
X/W		CAUTION!
X/W		If the application name specified in <i>appliname</i> corresponds to the primary
X/W		application name (generated in MAX APPLINAME=) then for
X/W		SIGNON-TAC= you may specify KDCSGNTC (standard sign-on service),
X/W		*NONE or leave it blank.
X/W	T-PROT=	Address formats of the T-selectors in the transport address.
X/W		You can specify the following address formats for T-PROT.
X/W	SOCKET	Communication is done via the socket interface.
X/W		No other address specifications are required other than T-PROT=SOCKET,
X/W		LISTENER-PORT and <i>appliname</i> .
X/W		You will find more information on SOCKET in section "Providing the address information for clients of type SOCKET" on page 153 .
X/W	RFC1006	Address format RFC1006
X/W		You will find more information on the RFC1006 address format in the "CMX
X/W		User Guide" or in the PCMX online help system.
X/W		Default: RFC1006
X/W	TSEL-FORMAT=	Format identifier of the T-selectors to be created from <i>appliname</i> .
X/W		The format identifier specifies the encoding of the T-selector in the transport
X/W		protocol. You will find more information in the "CMX User Guide" and in the
X/W		PCMX online help system.
X/W	T	TRANSDATA format (coding in EBCDIC)
X/W		In this case <i>appliname</i> must be exactly 8 characters long and must not
X/W		include lowercase letters.
X/W	E	EBCDIC format
X/W	A	ASCII format
X/W		Default:
X/W		T if the character set of <i>appliname</i> matches to the TRANSDATA format
X/W		E in all other cases
X/W		It is recommended for TNS-less operation via RFC1006 to explicitly specify
X/W		a value for TSEL-FORMAT.

CLUSTER – Define global properties of a UTM cluster application

The CLUSTER statement is used to configure a UTM cluster application. The operands of the CLUSTER control statement can be split over several CLUSTER statements.

If you specify the same operand in several CLUSTER statements, the first specification is taken as the valid one. No message is issued. Every mandatory operand must be specified once.

If a cluster statement is specified, you must also specify at least two CLUSTER-NODE statements. If a CLUSTER statement is specified, KDCDEF implicitly generates a BCAMAPPL entry with the BCAMAPPL name specified in the CLUSTER statement.



The effect of the CLUSTER statement depends on the value specified in the OPTION statement, see [section “OPTION - manage the KDCDEF run” on page 416](#).

If you change specifications in the CLUSTER statement or the CLUSTER-NODE statements with a new generation, you must create new UTM cluster files and a new KDCFILE (OPTION GEN=CLUSTER, KDCFILE) and use this file to apply the changes.

Exception:

The size of the cluster page pool can be increased during operation, i.e. without it being necessary to generate new UTM cluster files. When this is done, the number of cluster page pool files must not be changed.

```

CLUSTER    CLUSTER-FILEBASE = cluster_filebase
           ,BCAMAPPL = cluster_aplname
           ,USER-FILEBASE = user_filebase
           [ ,ABORT-BOUND-SERVICE = { NO | YES }
           [ ,CHECK-ALIVE-TIMER-SEC = time ]
           [ ,COMMUNICATION-REPLY-TIMER-SEC = time ]
           [ ,COMMUNICATION-RETRY-NUMBER = number ]
           [ ,DEADLOCK-PREVENTION = { NO | YES }
           [ ,EMERGENCY-CMD = command_string1 ]
           [ ,FAILURE-CMD = command_string2 ]
           [ ,FILE-LOCK-RETRY = number ]
           [ ,FILE-LOCK-TIMER-SEC = time ]
           [ ,PGPOOL=( number, warnlevel ) ]
           [ ,PGPOOLFS=number ]
           [ ,RESTART-TIMER-SEC = time ]

```

B***Additional operand in BS2000/OSD*****B**

```
[ ,IMPORT-USER-LOCALES = { NO | YES }
```

X/W***Additional operands in Unix systems and Windows systems*****X/W**

```
,LISTENER-PORT = port_number
```

X/W

```
[ ,LISTENER-ID=number ]
```

Mandatory operands



The operands CLUSTER-FILEBASE, BCAMAPPL, LISTENER-PORT (Unix systems and Windows systems) and USER-FILEBASE must always be specified. The specifications in the OPTION statement determine whether and in what way these operands are then evaluated.

CLUSTER-FILEBASE=cluster_filebase

Name prefix or directory for the UTM cluster files. Some of the UTM cluster files are generated by KDCDEF (see list below) while others are not generated until runtime.

The operand CLUSTER-FILEBASE is only evaluated if GEN=CLUSTER or GEN=(CLUSTER,...) is specified in the OPTION statement. In this case, KDCDEF generates the following files:

- the cluster configuration file
- the cluster user file
- the cluster page pool files.
- the cluster GSSB file
- the cluster ULS file

In this case, these files must not already exist.

Mandatory operand.

B

BS2000/OSD:

B

The UTM cluster files are created by KDCDEF under the file name *cluster_filebase.UTM-C.xxxx* where *xxxx* is file-specific, see [page 67ff](#).

B

These files can be renamed and/or copied to a different location to operate the UTM cluster application. If this is done, the suffix *.UTM-C.xxxx* must be retained. When an application is started, the valid name of the filebase must be specified in the start parameters. The name can be up to 42 characters in length and must comply with the syntax for file names. If you specify the name of the filebase without the catalog ID and user ID, you nevertheless still have to take account of the lengths of these. See also the section [“BS2000/OSD:” on page 46](#).

B

B

B

B

B

B

B

B

B

X/W

Unix systems / Windows systems:

X/W

cluster_filebase defines the directory in which the UTM cluster files are to be stored. The directory must be created before the KDCDEF run.

X/W

X/W

The UTM cluster files are created under the file names *UTM-C.xxxx*, where *xxxx* is file-specific, see [page 67ff](#).

X/W

X/W
X/W
X/W
X/W

The files can be copied to a different directory to operate the UTM cluster application. Specify the name which is then valid in the start parameters when the application is started. The name can be up to 42 characters in length and must comply with the syntax for file names.

BCAMAPPL=cluster_aplname

Name of the communication end point for cluster-internal communication.

The name specified here must be different from the names specified for TRANSPORT-SELECTOR under MAX APPLNAME, in other BCAMAPPL statements or in ACCESS-POINT statements. In addition, the name specified here must not be used by other applications on the computers of the UTM cluster application as the name of a communication end point.

The name generated here must not be referenced in other statements (e.g. in the PTERM statement) as the BCAMAPPL name.

The name can be up to 8 characters in length.

Mandatory operand.

LISTENER-PORT=port_number

Port number for cluster-internal communication.

This operand specifies the port number on which the local application listens for external connection requests. Enter any port number between 1025 and 65535.

Note that the port number specified here must not be used anywhere else on the computers of the UTM cluster. The port number must also differ from the other port numbers used by this application. KDCDEF does not, however, check this.

Mandatory operand.

USER-FILEBASE=user_filebase

Name prefix or directory for the current cluster user file of a UTM cluster application. The operand USER-FILEBASE is only evaluated if GEN=KDCFILE, GEN=(KDCFILE,ROOTSRC) or GEN=ROOTSRC is specified in the OPTION statement.

- If GEN=KDCFILE or GEN=(KDCFILE,ROOTSRC), the cluster user file must exist under the name taken from *user_filebase*. KDCDEF evaluates the file and extends it if necessary. The cluster user file can already be open for the KDCDEF run of a running UTM cluster application.
- If GEN=ROOTSRC then the cluster user file may already exist but this is not mandatory. If it does exist then it is checked but not modified.

Mandatory operand.

B	<i>BS2000/OSD:</i>
B	The name can be up to 42 characters in length and must comply with the syntax for file names. If you specify the name of the filebase without the catalog ID and user ID, you nevertheless still have to take account of the lengths of these.
B	
B	
B	
X/W	<i>Unix systems / Windows systems:</i>
X/W	The name can be up to 42 characters in length and must comply with the syntax for file names.
X/W	

Optional operands

ABORT-BOUND-SERVICE

This parameter determines how UTM behaves when a user who has an open service in a node application signs on.

- NO If there is a node-bound service for a user on sign-on (see note), 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.
- Default in UTM-S applications.
- This value is not permitted in UTM-F applications.
- YES If when a user signs on at a node application, there is a node-bound 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.
- Default in UTM-F applications.



A service is node-bound if it

- has a job-receiver service
- or has terminated a SESAM transaction
- or is an inserted service resulting from service stacking.

In addition, a service associated with a user is node-bound as long as the user is signed-on at a node application.

CHECK-ALIVE-TIMER-SEC=time

Interval in seconds at which a node application of a UTM cluster application checks the availability of another node application.

Minimum value: 30

Maximum value: 3600

Default value: 600

COMMUNICATION-REPLY-TIMER-SEC=time

Time in seconds that a node application of a UTM cluster application waits for a response after sending a message to another node application.

If there is no response in the time specified here, it must be assumed that the other node application has failed. If you have selected a value greater than zero for **COMMUNICATION-RETRY-NUMBER**, it is only assumed that the other node application has failed after the number of retry attempts has been reached.

Minimum value: 1

Maximum value: 60

Default value: 10

COMMUNICATION-RETRY-NUMBER=number

Number of retry attempts to establish communication with another node application if this node application does not respond within the time specified under **COMMUNICATION-REPLY-TIMER**. If the monitored node application also fails to respond to any of the retry attempts, it is flagged as failed.

Minimum value: 0, i.e. no retry after a timeout.

Maximum value: 10

Default value: 1

DEADLOCK-PREVENTION=

In UTM cluster applications, information concerning locked data areas (GSSB, TLS, ULS) is stored in a file. Before a service waits at a locked data area, UTM can check whether the new wait situation might result in a deadlock. To do this, additional file I/Os are necessary.

This parameter specifies whether or not UTM performs additional checks in order to prevent deadlocks.

YES UTM performs additional checks of the GSSB, TLS and ULS data areas in order to prevent deadlocks.

NO 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. See also **MAX** statement, operand **RESWAIT=time1** ([page 394](#)).

Default: NO

In productive operation, it is advisable to set this parameter to YES only if timeouts occur frequently when accessing these data areas.

EMERGENCY-CMD=command_string1

This operand passes a command string containing a command to be executed.

The emergency command is called by openUTM if a failed node application was not restarted after the FAILURE script has been called and the restart timer (RESTART-TIMER-SEC parameter) has expired.

The emergency command is always executed on the computer of the monitoring node application.

The string passed here is not parsed by KDCDEF.

command_string1 can be up to 200 characters in length. The way in which the emergency command is specified depends on the operating system.



When openUTM is installed, platform specific templates are supplied with the name

UTM-C.EMERGENCY or utm-c.emergency.

B

BS2000/OSD:

B

The name of a BS2000 SDF procedure. Specify the procedure as described under the SDF command ENTER-PROCEDURE (see the "BS2000/OSD-BC Commands" manual). The SDF procedure called must have 4 positional parameters. %s must be specified as a placeholder for each of the 4 procedure operands.

B

B

B

B

B

Example:

```
EMERGENCY-CMD = 'FROM-FILE=*LIB(SYSLIB.EXAMPLE,
UTM-C.EMERGENCY),PROC-PAR=(%s,%s,%s,%s)'
```

B

B

B

You can also specify further operands of the ENTER-PROCEDURE command.

B

B

Example:

```
EMERGENCY-CMD = 'FROM-FILE=*LIB(SYSLIB.EXAMPLE,EMERGENCY),
PROC-PAR=(%s,%s,%s,%s), LOGGING=*YES, JOBCLASS=<jobclass>
```

B

X

Unix systems:

X

The name of a Unix shell script. You must specify the fully qualified name.

X

Example:

```
EMERGENCY-CMD = '/opt/lib/utm61a00/32/shsc/utm-c.emergency'
```

X

W

Windows systems:

W

The name of a Windows command script. You must specify the fully qualified name.

W

W

Example:

```
EMERGENCY-CMD = 'c:\utm61-shsc-utm-c.emergency.cmd'
```

W

Calling the procedure or script command_string1 during an application run

Four arguments are passed to the procedure or script *command_string1*. These identify the failed cluster node and allow corrective measures to be initiated.

The arguments are passed in the following sequence:

APPLICATIONNAME Name of the UTM application.

FILEBASE Filebase name of the KDCFILE of the failed node application.

HOSTNAME Host name of the failed node.

VIRTUALHOST Virtual host name of the failed node.

The return code of the procedure or script is not evaluated.

B

BS2000/OSD:

B

In BS2000/OSD, the configured procedures are called with the ENTER-PROCEDURE command.

B

B

The BS2000 procedure called must have 4 positional parameters.

B

The %s placeholders are replaced by the values of APPLICATIONNAME, FILEBASE, HOSTNAME and VIRTUALHOST in this sequence when the procedure is called.

B

B

X

Unix systems:

X

The generated script *command_string1* is started as a background process.

X

The script *command_string1* is called with the following arguments:

X

APPLICATIONNAME FILEBASE HOSTNAME VIRTUALHOST

W

Windows systems:

W

The command script *command string* is called with the Windows command START without waiting for it to terminate.

W

W

It is called with the following arguments:

W

APPLICATIONNAME FILEBASE HOSTNAME VIRTUALHOST

FAILURE-CMD=command-string2

This operand passes a command string containing a command to be executed.

command_string2 can be up to 200 characters in length. The way in which the failure command is specified depends on the operating system.

The failure command is called by openUTM if a node application terminates abnormally or if failure of a node application is detected. A user can use the failure command to restart the failed node application, for instance.

The failure command is always executed on the computer of the monitoring node application.

Otherwise, the syntax and call method for FAILURE-CMD are identical to the syntax and call method of EMERGENCY-CMD (see [page 305](#)).



When openUTM is installed, platform specific templates are supplied with the name

UTM-C.FAILURE or utm-c.failure.

FILE-LOCK-RETRY=number

Number of retries for a lock request for a file that is global to the cluster if the lock was not assigned in the time specified in FILE-LOCK-TIMER-SEC.

Minimum value: 1

Maximum value: 10

Default value: 1

FILE-LOCK-TIMER-SEC=time

Maximum time in seconds that a node application of a UTM cluster application waits for a lock to be assigned to a file that is global to the cluster.

Minimum value: 10

Maximum value: 60

Default value: 30

X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W

LISTENER-ID=number

This parameter is used to select a network process for internal cluster communication if multi-threaded network connections have been generated (MAX ...,NET-ACCESS=MULTI-THREADED). In this case, it is possible to manage multiple thread connections in a network process, i.e. the communications of all ACCESS-POINTS and BCAMAPPLs with T-PROT≠SOCKET and the same LISTENER-ID are handled using the same network process.

X/W
X/W
X/W

Minimum value: 0

Maximum value: 32767

Default value: 0

PGPOOL=(number, warnlevel)

Specifies the size of the cluster page pool and the warning level for cluster page pool utilization. The cluster page pool stores the GSSB, ULS and the service data of users (USER statement) who are generated with RESTART=YES.

The cluster page pool can be extended during cluster operation while leaving the number of files unchanged, see the applicable openUTM manual "Using openUTM Applications".

number Size of the cluster page pool in 4K pages.

For each generated node, at least 500 4K pages are needed in the cluster page pool.

Default: 10,000 or the minimum size

Minimum value: $500 * \text{number of cluster nodes}$

Maximum value: $16777215 - (2 * \text{number in CLUSTER PGPOOLFS})$

If the value specified here is smaller than the minimum value that UTM calculates from the number of generated nodes and the length generated in MAX RECBUF=length, then UTM increases *number* to the minimum size.

warning level Percentage value specifying the cluster page pool utilization level at which a warning (message K041) is output.

Default: 80

Minimum value: 60

Maximum value: 99

Please note that the messages indicating that cluster page pool utilization has risen above or fallen below the warning level are only output for the node application that triggers the associated change in state. In contrast, all running node applications are affected by a potential cluster page pool bottleneck.

PGPOOLFS=number

Number of files over which the user data is to be distributed in the cluster page pool.

The cluster page pool files are created using the cluster filebase that is specified in the CLUSTER-FILEBASE operand. They are given the suffixes CP01, CP02, CP10.

In addition, KDCDEF always creates a file with the suffix CPMD which is used to manage the cluster page pool and does not contain any user data.

Default: 1

Minimum value: 1

Maximum value: 10

B	IMPORT-USER-LOCALES=	
B	YES	KDCDEF takes over the specifications for user locales into the cluster user file from the generation statements for users.
B		
B	NO	KDCDEF does not take over the specifications for user locales into the cluster user file from the generation statements for users already contained in the cluster user file.
B		
B		Default value: NO
	RESTART-TIMER-SEC=time	
		Maximum time in seconds that a node application requires for a warm start after a failure.
		After a failure has been detected and the failure command for a failed node application has been called, the monitoring node application starts a timer with the time specified here. If the failed node application is not available after this time has expired, the emergency command is started for the failed node application.
		If a value of 0 is specified, no timer is set for monitoring the restart of the failed node application.
		Minimum value: 0, i.e. restart of the application is not monitored.
		Maximum value: 3600
		Default value: 0

CLUSTER-NODE – Define a node application of a UTM cluster application

You use the CLUSTER-NODE statement to configure a node application of a UTM cluster application.

B An XCS cluster in BS2000/OSD supports a maximum of 16 node applications started simultaneously.

X/W In Unix systems and Windows systems, you can start up to 32 node applications simultaneously.

You are allowed to specify the CLUSTER-NODE statement up to 32 times for each UTM cluster application.

You must specify at least two CLUSTER-NODE statements if you want to generate a UTM cluster application. A CLUSTER statement must also be generated if you have specified a CLUSTER-NODE statement.



If you change specifications in the CLUSTER statement or the CLUSTER-NODE statements with a new generation, you must create a new cluster configuration file (OPTION GEN=CLUSTER) and use this file to apply the changes.

```
CLUSTER-NODE      FILEBASE = node_filebase
                  ,HOSTNAME = host_name
                  [ ,VIRTUAL-HOST = virtual_host_name]
```

B *Additional operand in BS2000/OSD*

B [,CATID = <catid_A>

FILEBASE = node_filebase

Base name of the KDCFILE, the user log file and the system log file SYSLOG for this node application. When a node application is started, the UTM system files are expected under the name specified here. The KDCFILE must be accessible from all node applications.

This operand replaces the FILEBASE start parameter in a standalone UTM application.

The base names of the CLUSTER-NODE statements must differ from each other. The same restrictions apply as for MAX KDCFILE=*filebase*.

Mandatory operand.

B	<i>BS2000/OSD:</i>
B	When a node application of a UTM cluster application is started, the UTM system files for this node application are expected under the name specified here. The <i>node_filebase</i> can contain a BS2000 user ID, but it must be specified without CATID. You must specify CATIDs using the CATID operand. The name can be up to 42 characters long – including CATID and USERID. See also the section “ BS2000/OSD: ” on page 46.
B	
B	
B	
B	
B	
X/W	<i>Unix systems / Windows systems:</i>
X/W	<i>node_filebase</i> identifies the directory containing the KDCFILE and all the files of the application when a node application of a UTM cluster application is started. The name specified here must identify the same directory from the perspective of all the computers of the cluster . The name can be up to 27 characters in length.
X/W	
X/W	
X/W	
X/W	
HOSTNAME=host_name	
Host name of this node. Specify the primary name of this host.	
The name can be up to 8 characters in length.	
The host names of the CLUSTER-NODE statements must differ from each other. Host names that only differ in terms of case are regarded as identical.	
X	In the case of Unix systems, you must specify the name of the computer that is output by the command <i>uname -n</i> .
X	
W	In Windows systems, you must specify the name of the computer that is entered in the Control Panel.
W	
Mandatory operand.	
VIRTUAL-HOST=virtual_host_name	
Has the same function as the MAX HOSTNAME parameter with UTM cluster applications. You are not allowed to specify the MAX HOSTNAME parameter in UTM cluster applications.	
B	<i>BS2000/OSD:</i>
B	Name of the virtual host on which the node application is to run from the perspective of BCAM.
B	
B	Default value: 8 blanks, i.e. the node application runs under the name of the real host.
B	

X/W

Unix systems / Windows systems:

X/W

X/W

X/W

X/W

X/W

X/W

HOSTNAME allows the sender address for network connections established from this node application to be specified. 8 blanks is the default. This means that the default sender address of the transport system is used when connections are established. This function is required in a cluster if the relocatable IP address is to be used as the sender address instead of the static IP address when establishing a connection.

B

CATID=(catid_A)

B

B

This operand specifies which which catalog ID the KDCFILE files are assigned to.

B

B

B

B

This operand replaces the CATID start parameter in a standalone UTM application. When a node application of a UTM cluster application is started, the UTM system files for this node application are expected in the catalogs specified here.

B

B

Please note that in UTM cluster applications, only single-file operation of the KDCFILE is permitted.

B

Default: No CATID

CON - define a connection for distributed processing based on LU6.1

The CON statement allows you to define a transport connection between the local UTM application and a partner application. It also assigns an LPAP partner to the real partner application, i.e. the logical access point of the partner application in the local application. You must define the LPAP partner in an LPAP statement (see [page 344](#)).

By issuing several CON statement for the same partner application, you can also define parallel transport connections.



For more information on generating LU6.1 connections see [section “Distributed processing via the LU6.1 protocol” on page 76](#).

When generating the CON, PTERM and MUX statements, please note that the name triplet (*appliname* or *ptermname*, *processername*, *local_appliname*) must be unique within the generation run.

Example

If a PTERM statement has already been generated with

```
PTERM partner_name1, PRONAM=processername1,
```

you cannot generate a CON statement with

```
CON partner_name1, PRONAM=processername1,
```

but you can enter

```
CON partner_name1, PRONAM=processername1, BCAMAPPL=local_appliname1,
```

provided *local_appliname1* is not identical to the primary UTM application name.

B
B

The statements MUX *partner_name1* . . . and CON *partner_name1* are also mutually exclusive.

```

CON_    remote_appliname
        [ ,BCMAPPL=local_appliname ]
        [ ,LPAP=lpapname ]
        ,PRONAM=processorname           only mandatory under BS2000/OSD
        [ ,TERMN=termn_id ]

```

X/W

Unix system and Windows system specific operands

X/W

```
[ ,LISTENER-PORT=number ]
```

X/W

```
[ ,T-PROT=RFC1006 ]
```

X/W

```
[ ,TSEL-FORMAT={ T | E | A } ]
```

remote_appliname

Name of the partner application with which you wish to communicate via the logical connection.

remote_appliname can be up to eight characters in length. Permitted characters are capital letters, numbers and the characters \$, # and @. Hyphens are not allowed in names. The first letter must be a capital letter. If lowercase letters are used in a name, you must enter it in single quotes ('...').

remote_appliname is a mandatory specification.

B

BS2000/OSD:

B

remote_appliname can be either the BCAM name of a UTM partner application (in the case of a homogenous link) or the name of a TRANSIT application (in the case of a heterogeneous link).

B

The first letter must be uppercase.

B

X/W

Unix systems, Windows systems:

X/W

You must specify the T-selector that the partner application uses to sign on to the transport system for *remote_appliname*.

X/W

The first character must be a letter.

X/W

BCAMAPPL=local_applname

A name for the local application, as defined in the MAX or BCAMAPPL control statement. A BCAMAPPL name may not be specified for which T-PROT=SOCKET is generated.

X/W
X/W

Under Unix systems and Windows systems this name must not begin with a '\$'.

The BCAMAPPL name specified in the CLUSTER statement is not permitted here.

Default:

If nothing is specified, then the primary application name defined in MAX ...,APPLNAME= is used.

X/W
X/W
X/W

LISTENER-PORT=number

Port number of the partner application if the connection to the partner application is established via TCP/IP.

X/W

Permitted values: 102 and 1025 - 32767

X/W
X/W
X/W

Default: 0 (no port number)

If OPTION CHECK-RFC1006=YES, then a port number must be entered for LISTENER-PORT.

LPAP=lpapname

Name of the LPAP partner of the partner application with which the connection is to be established. The name of the LPAP partner via which the partner application signs on must be defined using the statement LPAP *lpapname*.

By issuing several CON statements with the same *lpapname*, you can establish parallel connections to the partner application.

Please note, however, that these parallel connections lead to the same partner application (*remote_applname* and *processorname*).

Mandatory parameter

PRONAM={ processorname | C'processorname' }
 Name of the host partner up to 8 characters in length
 If the *processorname* contains lower case letters or special characters it must be entered as a character string using C'...'

Mandatory operand for BS2000/OSD

B

BS2000/OSD:

B

For *processorname* you must specify the name of the processor on which the partner application *remote_appliname* runs. This is either the name of a Unix system, Windows system or BS2000 system, or the TRANSDATA name of a communication computer or front-end processor in the case of a heterogeneous link. This name is defined during generation of the network. Please consult your network administrator.

B

B

B

B

B

B

B

B

B

PRONAM needs not to be specified if a default value for this operand is defined beforehand using the DEFAULT statement.

X/W

Unix systems and Windows systems:

X/W

There are two options for specifying the *processorname*:

X/W

– You enter the real host name under which the IP address of the partner computer is entered in the name service of the local system (e.g. the hosts file). You must not specify an alias of the computer.

X/W

X/W

X/W

– You enter the UTM host name of the partner computer.

X/W

This is only possible when you have set the UTM_NET_HOSTNAME environment variable and have specified the UTM host name in the conversion file (see the [section “Using mapped host names \(Unix systems and Window systems\)” on page 122](#)).

X/W

X/W

X/W

X/W

X/W

Default: 8 blanks

TERMN=termn_id

Identifier up to two characters in length, which indicates the type of communication partner. *termn_id* is not queried by openUTM, but is used by the user when querying or grouping terminal types, for example. *termn_id* is entered in the KB header for job-receiving services, i.e. for services started by a partner application in the local application.

Default: A4

X/W

T-PROT=

Address format with which the partner application signs on to the transport system. The following address formats are explained in the CMX User Guide and in the PCMX online help system.

X/W

X/W

X/W

X/W	RFC1006	Address format RFC1006
X/W		Default: RFC1006
X/W	TSEL-FORMAT=	
X/W		Format identifier of the T-selector. The format indicator specifies the encoding of the T-selectors in the transport protocol. You will find more information in the CMX User Guide and in the PCMX online help system.
X/W	T	TRANSDATA format (encoded in EBCDIC)
X/W	E	EBCDIC character format
X/W	A	ASCII character format
X/W		Default:
X/W		T if the character set of the T-selector corresponds to the TRANSDATA format.
X/W		E in all other cases
X/W		It is recommended to specify a value explicitly for TSEL-FORMAT for operation via RFC1006.

X/W **The address of a partner application of a UTM application under Unix systems and Windows systems**

X/W In order to be able to establish a connection to a partner application, the UTM application must know the address of the partner application. You can enter the address using the following operands:

- X/W – *remote_applname* (address of the partner application in the partner processor)
- X/W – PRONAM (real host name or UTM host name of the partner processor)
- X/W – LISTENER-PORT (port number for RFC1006)
- X/W – T-PROT (the transport protocol used)
- X/W – TSEL-FORMAT (format indicator of the T-selector)

X/W See [section “Providing address information for the CMX transport system \(Unix systems and Windows systems\)”](#) on page 110.

CREATE-CONTROL-STATEMENTS - Create KDCDEF control statements

When regenerating your application, inverse KDCDEF allows you to retain UTM objects in the configuration which were entered dynamically during runtime. Further information can be found in [section “Inverse KDCDEF” on page 268](#).

As the first statement in the KDCDEF run, CREATE-CONTROL-STATEMENTS generates KDCDEF control statements for the UTM objects entered dynamically, and outputs them to the file *control_statements_file* (inverse KDCDEF). During the same KDCDEF run, you can use *control_statements_file* as the basis for generation by defining it as an input file using the statement `OPTION ...,DATA=control_statements_file`.

If the file *control_statements_file* containing the generated KDCDEF control statements is to be processed at a later point in time in a KDCDEF run, you must conclude the control statements in the file using the END statement.

Inverse KDCDEF can generate control statements for UTM objects of type TAC, PROGRAM, PTERM, LTERM, USER, CON, LTAC, LSES and KSET.

If, when performing regeneration with KDCDEF, no user is specified with `PTYPE=APPLI`, `SOCKET` or `UPIC-R` for a PTERM or TPOOL statement in an assigned LTERM definition, KDCDEF implicitly generates a user with the LTERM name. However, for users generated in this way, inverse KDCDEF **neither** creates USER statements **nor** adds the user name in the `USER=` operand in the LTERM statement.

It does not transfer UTM objects to the file *control_statements_file*, which were marked for deletion by administration using `KC_DELETE_OBJECT`. Following a KDCDEF run in which *control_statements_file* is defined as an input file, the names of the deleted UTM objects are no longer reserved.

You can start KDCDEF for inverse KDCDEF with at least one CREATE-CONTROL-STATEMENTS statement and without any further KDCDEF control statements.



If you upgrade to a new version, the KDCDEF control statements must first be created in the previous version before being processed in a later version by the KDCDEF generation tool.

```
CREATE-CONTROL-STATEMENTS_ { *ALL | CON | DEVICE | KSET | LSES | LTAC |
                             PROGRAM | USER }
                             ,FROM-FILE=kdcfile
                             ,TO-FILE=control_statements_file
                             [ ,MODE={ CREATE | EXTEND } ]
```

- *ALL** KDCDEF control statements are generated for the following object types:
- KSET
 - LSES
 - LTAC
 - TAC
 - CON
 - PROGRAM
 - PTERM
 - LTERM
 - USER
- They cannot be created for other object types.
- CON** This creates KDCDEF control statements for the transport connections to remote applications.
- DEVICE** KDCDEF control statements are generated for LTERM partners, clients and printers, i.e. for the following object types:
- PTERM
 - LTERM
- KSET** KDCDEF control statements are generated for key sets, i.e. for objects of type KSET.
- LSES** This creates KDCDEF control statements for the assignment of session names.
- LTAC** KDCDEF control statements are generated for transaction codes via which service programs in partner applications are started. These are objects of the type LTAC.
- PROGRAM** KDCDEF control statements are generated for programs, service exits transaction codes and TAC queues , i.e. for the following object types:
- TAC
 - PROGRAM

USER KDCDEF control statements are generated for user IDs, i.e. for objects of type USER.



Please note that passwords cannot be reconstructed. In the case of user IDs with passwords, statements are created with the following format:
 USER username, PASS=*RANDOM, . . .

In the case of standalone applications, you must use the KDCUPD tool to transfer the passwords to the new KDCFILE after the KDCDEF run has finished. This is also possible for the UTM-F generation variant.

FROM-FILE=kdcfile

Name of the KDCFILE from which the control statements are to be generated.



The openUTM version of the KDCFILE must match that of the KDCDEF generation tool.

TO-FILE=control_statements_file

The generated KDCDEF control statements are written to the file specified in *control_statements_file*. For *control_statements_file*, you must enter a valid file name. *control_statements_file* can be defined as an input file for the KDCDEF run using the statement OPTION ...,DATA=*control_statements_file*.

MODE= Write mode of the file containing the generated KDCDEF control statements

CREATE

The file specified in *control_statements_file* is created.

B
B
B

Under BS2000/OSD the file is created as a SAM file. If a file with the same name already exists, this must be a SAM file. This SAM file is then overwritten.

X/W
X/W

If a file with the same name already exists under Unix systems or Windows systems, it is overwritten.

EXTEND

The generated control statements are appended to the existing *control_statements_file*. If this file does not exist, it is created.

DATABASE - define the database system (BS2000/OSD)

- B The DATABASE control statement allows you to define the database systems with which
- B the UTM application is to coordinate.
- B Each database system must be defined in a separate DATABASE statement. By issuing
- B several DATABASE statements for the same database system, you can assign several entry
- B names to that database system.
- B The DATABASE statement can be issued several times. It is thus possible to define up to
- B two (in a special release, up to eight) different database systems.

```

B DATABASE_ [ ENTRY=entryname ]
B           [ ,LIB=omlname | LOGICAL-ID(logical-id) } ]
B           [ ,TYPE={ UDS | SESAM | PRISMA | LEASY | DB | XA } ]


```

ENTRY=entryname

Entry name of the database. The following default values apply:

- B \$UNIBASE if TYPE = UDS
- B SESAM if TYPE = SESAM
- B PRISCON if TYPE = PRISMA
- B LEASY if TYPE = LEASY
- B DB if TYPE = DB

- B When generating the XA connection with TYPE=X A in openUTM under
- B BS2000/OSD, the name of the XA switch as it is provided by the database
- B system **must** be specified with the ENTRY parameter. It is possible to
- B generate several XY switches in the DATABASE statement.

- B  A database connection to Oracle must be generated with TYPE =
- B XA.

- B Other entry names (e.g. SQLUDS for UDS/SQL) can be found in the
- B manuals for the respective database systems.


B LIB= Specifies library from which the connection module for the database system
B is dynamically loaded.

B omName
B OML name which the connection module for the database system is to be
B loaded dynamically. *omName* can be up to 54 characters in length.

B LOGICAL-ID(logical-id)
B Specifies that a search is to be made for the connection module in the IMON
B installation path for the database system and that the module is to be loaded
B from there.

B *logical-id* is a name up to 15 characters long. Currently it may be specified
B only for SESAM/SQL and UDS/SQL; it is SYSLNK for both database
B systems, refer also to the notes on [page 322](#).

B If you do not specify LIB= , then LIB= is set to TASKLIB. This does not corre-
B spond to the SET-TASKLIB command, rather a library named TASKLIB
B must exist. Dynamic loading of the connection module from the library
B assigned with SYSDIR-TASKLIB is not supported.

B  During the dynamic load, the DBL searches for the connection module first
B in the library that you specified in LIB= . If this library does not exist, then
B the DBL aborts the search. If the library exists but the connection module is
B not found there, then the DBL searches the alternative libraries. These
B libraries are the libraries that have been assigned a file chain name
B BLSLIB nn ($0 \leq nn \leq 99$).

B If several DATABASE statements are issued with the same TYPE in order
B to generate a number of entries for the same database, the connection
B module is loaded from the library specified in the LIB operand of the first
B DATABASE statement with the relevant TYPE.

B *Notes on using LOGICAL-ID*

B – LIB=LOGICAL-ID(*logical-id*) may be specified only if the database
B system was correctly installed with IMON. If the database system was
B not installed with IMON, you must either statically link the connection
B module (without the LIB= operand) or you must specify LIB=*omName*.

B – Using LOGICAL-ID(*logical-id*) instead of *omName* has the advantage
B that the openUTM application is then independent of the installation
B paths and library names of the database system.

B – If you specify LIB=LOGICAL-ID(SYSLNK) and if several product
B versions are installed, the most recent version is used by default.

B
B
B
B

- If you do not want the most recent version to be loaded, you must either specify the library of a less recent version using `LIB=omlname` or you must assign the version before starting the openUTM application (using the `SELECT-PRODUCT-VERSION` command of IMON).

B
B
B

- If an error occurs when searching for the connection module in the IMON installation path, application start is aborted and the error is logged to SYSOUT.

B

TYPE=

This identifies the database system.

B
B
B

With `TYPE=DB` you can also connect to database systems other than those named above. This is only possible when the database system supports the IUTMDB interface.


B

Default: UDS

DEFAULT - define default values (BS2000/OSD)

B The DEFAULT control statement allows you to define default values for the operands of a
 B KDCDEF control statement. A default operand value set using DEFAULT applies until the
 B next DEFAULT statement is issued for the same operand in the same control statement. If
 B you subsequently wish to reset the default value to the UTM standard setting, you must
 B reassign this standard setting using the DEFAULT statement. If this is not possible (e.g.
 B FORMAT = blanks), then the default value is set in the (STD) or *STD entry.

B Statement-specific default values offer the following advantage:
 B If you issue a control statement several times (e.g. PTERM), there is no need to specify the
 B same operand values over and over again in each statement (e.g. the processor name in
 B PRONAM).

B  When porting BS2000/OSD openUTM applications to Unix systems or Windows
 B systems, please note that the DEFAULT statement is not supported by openUTM
 B under Unix systems and Windows systems.

B DEFAULT_ control-statement_operand [,operand] [,...]

B control-statement
 B KDCDEF control statement for which new default values are to be defined
 B in this DEFAULT statement. The following operands are dependent on this
 B control statement, and apply only for this control statement class. Please
 B note that the PROGRAM and AREA statements form a **single** class, i.e.
 B modified default values of the PROGRAM statement also apply for the other
 B statements in this class.

B You must insert at least one blank between *control_statement* and the
 B following operands. The table on the next page shows the control state-
 B ments that can be specified here.

B operand ,... One or more operands of the KDCDEF control statement *control_statement*.
 B Each operand is separated by a comma. The table on the next page shows
 B the operands permitted for *control_statement*.

	Permitted control statements	Permitted operands
B B B B	CON	BCAMAPPL={ <i>local_appliname</i> (STD)} LPAP={ <i>lpapname</i> (STD)} PRONAM={ <i>processorname</i> C' <i>processorname</i> '} TERMN= <i>termn_id</i>
B B B B	LPAP	NETPRIO= <i>netprio</i> QLEV= <i>queue_level</i> STATUS={ONIOFF} SESCHA= <i>sescha_name</i>
B	LSES	LPAP= <i>sessionname</i>
B B B B B	LTAC	LPAP= <i>lpapname</i> LTACUNIT=% <i>_ltacunit</i> STATUS={ONIOFF} TYPE={DIA} WAITTIME=(<i>time1,time2</i>)
B B B B B B B B B B	LTERM	ANNOAMSG={YIN} FORMAT={ <i>formatname</i> (STD)} KERBEROS-DIALOG={YES NO} LOCALE={ ([<i>lang_id</i>], [<i>terr_id</i>],[<i>ccsname</i>]) *STD} NETPRIO= <i>netprio</i> PLEV= <i>print_level_number</i> QAMSG={YINI (STD)} QLEV= <i>queue_level_number</i> RESTART={YESINO} STATUS={ONIOFF} USAGE={DIO}
B B B	LOAD-MODULE	LIB= <i>libname</i> LOAD-MODE= <i>loadmode</i> VERSION= <i>version</i>
B B	OSI-CON	ACTIVE={YESINO} LOCAL-ACCESS-POINT= <i>access-point_name</i>
B B B B B	OSI-LPAP	APPLICATION-CONTEXT= <i>application_context</i> IDLETIME= <i>time</i> QLEV= <i>queue_level_number</i> STATUS={ONIOFF} TERMN= <i>termn_id</i>
B B	PROGRAM	COMP= <i>compiler</i> LOAD-MODULE={ <i>lmodname</i> *STD}

	Permitted control statements	Permitted operands
B B B B B B B B B B B	PTERM	BCAMAPPL= <i>local_appliname</i> CONNECT={AIN} ENCRYPTION-LEVEL={NONE 1 2 3 4 TRUSTED} IDLETIME= <i>time</i> MAP={USER SYSTEM SYS SYS1 SYS2 SYS3 SYS4} PRONAM={ <i>processorname</i> C' <i>processorname</i> ' *RSO} PROTOCOL={NISTATION} PTYPE={ <i>partnertyp</i> *RSO *ANY} STATUS={ONIOFF} TERMN={ <i>termn_idl</i> (STD)} USAGE={DIO} USP-HDR={ALL MSG NO}
B B B B B B	SESCHA	CONNECT={YIN} CONTWIN={YINI(STD)} DPN={ <i>instance_name</i> l(STD)} IDLETIME= <i>time</i> PLU={YINI(STD)} PACCNT= <i>number</i>
B B B B B B B B B B B B B B B B B B B	TAC	ADMIN={Y N} CALL={BOTH FIRST NEXT (STD)} DEAD-LETTER-Q={NO YES} ENCRYPTION-LEVEL={NONE 1 2 } EXIT={ <i>exit</i> (STD) } PGWT={NO YES} PROGRAM={ <i>program_name</i> l(STD)} QLEV= <i>queue_level_number</i> QMODE = {STD WRAP-AROUND} RUNPRIO= <i>priority</i> SATADM={NOIYES} SATSEL={BOTHISUCCIFAILINONE} STATUS={ON OFF HALT KEEP} TACCLASS={ <i>class</i> l(STD)} TACUNIT= <i>tacunit</i> TCBENTRY={ <i>name_of_tcbentry-statement</i> (STD)} TIME={ <i>time1</i> (<i>time1,time2</i>)} TYPE={D A Q}

	Permitted control statements	Permitted operands
B B B B B B B B B B B B B B	TPOOL	ANNOAMSG={ Y N } BCAMAPPL= <i>applname</i> ENCRYPTION-LEVEL={ NONE 1 2 3 4 TRUSTED } FORMAT={ <i>formatname</i> (STD)} IDLETIME= <i>time</i> KERBEROS-DIALOG={ YES NO } LOCALE={ ([<i>lang_id</i>], [<i>terr_id</i>],[<i>ccsname</i>]) *STD } MAP={ USER SYSTEM SYS SYS1 SYS2 SYS3 SYS4 } NETPRIO={ MEDIUM LOW } NUMBER= <i>number1</i> PRONAM={ <i>processorname</i> C' <i>processorname</i> ' *ANY } PROTOCOL={ N STATION } PTYPE={ <i>partnertyp</i> *ANY } QLEV= <i>queue_level_number</i> TERMN={ <i>termn_id</i> (STD) } USP-HDR={ ALL MSG NO }
B B B B B B B B B B	USER	FORMAT={ <i>formatname</i> (STD)} LOCALE={ ([<i>lang_id</i>], [<i>terr_id</i>],[<i>ccsname</i>]) *STD } PERMIT={ NONE ADMIN SATADM (ADMIN,SATADM) } PROTECT-PW=(<i>length,level_of_complexity,max_time,min_time</i>) QLEV= <i>queue_level_number</i> QMODE = { STD WRAP-AROUND } Q-READ-ACL = <i>keysetname</i> Q-WRITE-ACL = <i>keysetname</i> RESTART={ YES NO } SATSEL={ BOTH SUCC FAIL NONE } STATUS={ ON OFF }

EDIT - define edit options (BS2000/OSD)

B The EDIT control statement allows you to combine screen functions and screen output
 B properties in line mode (edit options) in groups known as edit profiles. It also enables you
 B to assign names to these edit profiles, which can then be used to address a set of edit
 B options from a program unit.

B The EDIT statement can be issued several times within a generation run. However, a
 B different name (*name* operand) must be specified in each EDIT statement.


B The edit profile names are specified in the KCMF field of the MPUT, MGET, DPUT, FPUT
 B and FGET calls at the programming interface, where a blank is entered as the format control
 B character.

B openUTM interprets the entries in the KCMF field as follows:

	No edit profiles generated	Edit profiles generated
B B B B	If a blank is entered as the format control character, openUTM ignores the remaining characters in the field.	If a blank is entered as the format control character, the remaining characters in the field must contain either the name of a valid edit profile or further blanks.

B A detailed description of the operands described below can be found in the TRANSDATA
 B TIAM User Guide. Further information on working with edit profiles can be found in the
 B openUTM manual “Programming Applications with KDCS”.

B	EDIT_ name
B	[,BELL={ <u>NO</u> YES }]
B	[,CCSNAME=ccsname]
B	[,HCOPY={ <u>NO</u> YES }]
B	[,HOM={ <u>NO</u> YES }]
B	[,IHDR={ <u>NO</u> YES }]
B	[,LOCIN={ <u>NO</u> YES }]
B	[,LOW={ <u>YES</u> NO }]
B	,MODE={ EXTEND INFO LINE PHYS TRANS }
B	[,NOLOG={ <u>NO</u> YES }]
B	[,OHDR={ <u>NO</u> YES }]
B	[,SAML={ <u>NO</u> YES }]
B	[,SPECIN={ C I <u>N</u> }]

B B	name	Alphanumeric name up to seven characters in length for the set of edit options to be defined.
B B	BELL=	This specifies whether or not an acoustic alarm is triggered on the terminal when a message is output.
B B B B B B B B B	CCSNAME=ccsname	<p>(coded character set name) Name of the character set (CCS name) used to format a message. This name can be up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). The character set must be compatible with an ISO character set supported by the terminal to which the message is directed. During generation, KDCDEF cannot check the validity of the CCS name under the BS2000 system or the compatibility condition.</p>
B B		A CCS name must not be assigned to the edit profile if the value TRANS (transparent mode) is defined for the MODE operand.
B B		If the edit profile is used to output messages to an RSO printer, only the CCSNAME= parameter of the edit profile is evaluated.
B B B	HCOPY=	<p>(hard copy) This specifies whether the output message is to be logged on a connected hardcopy printer in addition to being displayed on the terminal.</p>
B B B B	HOM=	<p>(homogeneous) This specifies whether the output message is to be output unstructured, i.e. in homogeneous format. If you enter NO here, the message is output in a structured format, i.e. in non-homogeneous format. In this case, a logical line is regarded as an output unit.</p>
B B B	IHDR=	<p>(input header) This specifies whether the header of the input message is to be transferred to the program unit.</p>
B B B B B	LOCIN=	<p>(local parameter input) This operand applies only for terminals that support local parameters (e.g. 9763). If you enter YES here, local attributes in the input message are forwarded to the user as logical control characters. If you enter NO here, local attributes are removed from the input message and are not forwarded. LOCIN=YES is permitted only if MODE=EXTEND.</p>
B B B B	LOW=	<p>(lowercase) This specifies whether lowercase letters are permitted in the input message transferred to the program unit. If you enter NO here, the system converts all lowercase letters into uppercase.</p>

B	MODE=	
B	EXTEND	(extended line mode) This specifies whether the message is to be output in extended line mode. If you enter MODE=EXTEND, the value YES is permitted only for the BELL, LOW, and LOCIN edit options. The value N must be entered for the SPECIN operand.
B		
B		
B	INFO	The message is to be output in a special information line (system line) without overwriting important data at the terminal.
B		
B		This specification is primarily intended for application programs that send "asynchronous" messages to terminals without knowing what is currently being displayed at the terminal. At terminals with a hardware display line (e.g. DSS 9749, 9750, 9763), the data is always output protected in a hardware system line; in all other cases, it is output in the same way as a normal line mode message.
B	LINE	(line mode) The message is to be output in line mode. It can be structured using logical control characters, and is formatted by the system. If you enter MODE=LINE, the value NO must be entered for IHDR, OHDR and LOCIN.
B		
B	PHYS	(physical mode) The message is to be output or read in physical mode, i.e. without being formatted by the system. If you enter MODE=PHYS, the value YES is permitted only for the IDHR, LOW and OHDR edit options. The value N must be entered for the SPECIN operand.
B		
B		This specification should not be used for messages output on a printer. Physical messages to a printer can only be implemented using a format exit.
B	TRANS	(transparent mode) The output message is to be transferred in transparent mode. If you enter MODE=TRANS, the value YES must not be specified for any other edit option.
B		
B		The value N must be entered for the SPECIN= operand. The CCSNAME= operand must not be specified.
B	NOLOG=	(no logical characters) This specifies how the system is to handle non-printable characters.
B		
B	YES	The logical control characters are not evaluated. All characters less than X'40' in EBCDIC code are replaced by alternate characters (SUB). Only printable characters are allowed through.
B		

B	NO	All logical control characters are evaluated. Special physical control characters are allowed through. All other characters less than X'40' are replaced by alternate characters (SUB). Printable characters are allowed through.
B		
B		
B		
B		Default: NO
B	OHDR=	(output header) This specifies whether the output message contains a header. The length of the message header + 1 must be entered in binary format in the first byte of the message.
B		
B		
B		
B	SAML=	(same line) This applies only for printer stations. If SAML=YES, the message is not preceded by a line feed. If SAML=NO applies, the message begins at the start of the next line.
B		
B	SPECIN=	(special input)
B	C	(confidential) This specifies whether the display of input data is to be suppressed on the terminal, thus protecting confidentiality.
B		
B		
B	I	(id-card) This specifies whether input data is to be entered via the ID card reader.
B		
B	N	(normal) The terminal requires normal input.
B		

EJECT - initiate a page feed in the log

The EJECT control statement allows you to initiate a page feed in the log. The EJECT line itself is not logged or counted.

EJECT_

END - terminate KDCDEF input

The END control statement identifies the end of the sequence of control statements, and is the last statement entered.

END_



If a file with `OPTION DATA=filename` is defined as a KDCDEF input file and contains an END statement, KDCDEF input is terminated as soon as this statement is processed.

EXIT - define event exits

The EXIT control statement allows you to define event exits, which are used in the application.

For the event exits FORMAT and INPUT, you may only specify a single EXIT statement for each KDCDEF run.

For the event exits START and SHUT, you may specify up to eight EXIT statements. However, the specifications for the PROGRAM= operand must differ for the EXIT statements.

When starting or terminating a UTM process, all of the programs defined as START or SHUT exits are called one after the other. The sequence of the EXIT statements in the KDCDEF run determines the sequence in which openUTM activates the START or SHUT exit program.

Further information on event exits can be found in the openUTM manual “Programming Applications with KDCS”.

B *Event exits under BS2000/OSD:*

B The event exits START, SHUT, INPUT and FORMAT must not be assigned to a load module generated with LOAD-MODULE LOAD-MODE=ONCALL.






The event services MSGTAC, BADTACS and SIGNON must be defined using the TAC statement.

```
EXIT_      PROGRAM=objectname
           ,USAGE={ START |
                   SHUT |
                   ( INPUT, { ALL | FORMMODE | LINEMODE | USERFORM1 } ) |
                   FORMAT1 }
```

B ¹ FORMAT and USERFORM are only permitted under BS2000/OSD

PROGRAM=name

Name of the program containing the functions to be executed for the event exit. A PROGRAM statement with this name (*objectname*) must be issued.

USAGE=	Type of event exit
START	Used as event exit START
SHUT	Used as event exit SHUT
INPUT	Used as event exit INPUT Additionally you must specify the type of INPUT exit:
ALL	Event exit INPUT, which handles messages of all format control characters as well as LINEMODE messages.
	If you specify ALL here, this is the only event exit INPUT of the application. Further INPUT event exits cannot be defined.
FORMMODE	Event exit INPUT for +, *, and #formats
LINEMODE	Event exit INPUT for LINEMODE messages
	USERFORM Event exit INPUT for -formats
	FORMAT Used as event exit FORMAT

FORMSYS - define the format handling system

The FORMSYS control statement allows you to define the format handling system. Only the first FORMSYS statement is evaluated.

```
FORMSYS_ [ TYPE=typ ]
B        [ ,ENTRY=entryname ]
B        [ ,LIB=omlname ]
```

TYPE=typ This identifies the format handling system.

B – Under BS2000/OSD at present, only the value FHS is supported.

B Default (BS2000/OSD): FHS


B ENTRY=entryname

B Entry name for the format handling system

B Default value: KDCFHS with TYPE=FHS

B LIB=omlname Designates the object module library (OML) from which the connection
B module for the format handling system is loaded. *omlname* can be up to
B 54 characters in length.

B If you do not specify LIB= , then LIB= is set to TASKLIB. This does not corre-
B spond to the SET-TASKLIB command, rather a library named TASKLIB
B must exist. Dynamic loading of the connection module from the library
B assigned with SYSDATA-TASKLIB is not supported.

B  During the dynamic load, the DBL searches for the connection module first
B in the library that you specified in LIB= . If this library does not exist, then
B the DBL aborts the search. The DBL does not abort the search if LIB was
B not specified but was preset to TASKLIB and no file with this name exists.
B If the library exists but the connection module is not found there, then the
B DBL searches the alternative libraries. These libraries are the libraries that
B have been assigned a file chain name BLSLIB nn ($0 \leq nn \leq 99$).

KSET - define a key set

The KSET control statement allows you to combine the key codes of an application, which were defined for data access control, to form a logical key set. You can specify several control statements for a single key set.

KDCDEF implicitly generates the KDCAPLKS key set, which by default contains all key codes.

```
KSET_      keysetname
           ,KEYS={ ( key1,key2,... keyn) | MASTER }
```

keysetname Name of the defined key set up to eight characters in length.

You can assign this key set

- to a user (in a USER statement on [page 534](#))
- to an LTERM partner (in a LTERM statement on [page 360](#))
- to a partner application (in an LPAP or OSI-LPAP statement on [page 345](#) or [page 432](#))
- to a TAC (in the TAC statement on [page 492](#))
- to an LTAC (in the LTAC statement on [page 349](#))
- to a TPOOL (in the TPOOL statement on [page 519](#) or [page 526](#))

After the connection has been established, the key set of the LTERM or (OSI-)LPAP partner assigned to the connection is available to the client or partner application. After signing on to the application, the key set of the user ID is available to the client or partner application.

The lock/key code and the access list concept are described in detail in the openUTM manual “Concepts und Functions”. An introduction to access control can be found in [section “Lock/key code concept” on page 219](#).

KEYS= Key or access codes of the key set *keysetname*

(*key1*,..., *keyn*)

List of numbers between 1 and the maximum value permitted by the application (MAX ...,KEYVALUE=*number*). These numbers correspond to the key codes contained in this key set.

A key or access code grants access to a resource secured with a lock code or an access list, provided the key code and lock code match or the access code is contained in the access list.

You can specify up to 60 key codes/access codes in each KSET statement. If a key set contains more than 60 key codes, you must issue another KSET statement with the same *keysetname*.

If you only specify one key code, you can omit the parentheses.

If you enter the value 0 for *key*, this is ignored by openUTM. No message is output.

MASTER The MASTER key set contains all the key codes/access codes of the application.

LOAD-MODULE - define a load module (BLS, BS2000/OSD)

- B The LOAD-MODULE control statement allows you to define the name, version and
B properties of load modules. If you use the BLS interface, this statement must be issued for
B all load modules that can be exchanged or loaded as independent units during the program
B run. Each load module must be defined in a separate LOAD-MODULE statement.
- B The load modules that can be processed with BLS are either LLMs (link and load modules)
B or OMs (object modules). However, it is recommended that the program components and
B data areas to be loaded dynamically are linked to LLMs (see the BLS manuals).
- B A load module can contain several program units and data areas, which are defined using
B PROGRAM or AREA statements. You can assign one or more PROGRAM and/or AREA
B statements to a single LOAD-MODULE statement. This takes place on the basis of the load
B module name *lmodname*, which must also be entered in the LOAD-MODULE operand of the
B PROGRAM or AREA statement. However, it is also possible to generate LOAD-MODULE
B statements without assigning a PROGRAM or AREA statement (e.g. load modules that
B contain parts of the runtime system of a programming language).
- B At least one LOAD-MODULE statement must be generated if the "program exchange"
B function (KDCAPPL PROGRAM=NEW) is to be used under BS2000/OSD.
- B When starting the UTM application, the load modules are loaded in accordance with the
B sequence of LOAD-MODULE statements and the value of the LOAD-MODE operand. The
B load sequence is as follows:
- B ● The basic part of the application, including all load modules linked in statically to the
B application program (LOAD-MODE=STATIC).
 - B ● All load modules loaded into a global common memory pool when starting the UTM
B application. These are generated with LOAD-MODULE LOAD-MODE=(POOL,
B *poolname*,...) and MPOOL *poolname*,SCOPE=GLOBAL.
B The common memory pools are loaded in accordance with the sequence of MPOOL
B statements in the generation run. Within a pool, the sequence of LOAD-MODULE state-
B ments that refer to this pool applies.
 - B ● All load modules loaded into a local common memory pool when starting the UTM appli-
B cation. These are generated with LOAD-MODULE LOAD-MODE=(POOL, *poolname*,...)
B and MPOOL *poolname*,SCOPE=GROUP.
B The pools are loaded in accordance with the sequence of MPOOL statements. Within
B a pool, the sequence of LOAD-MODULE statements that refer to this pool applies.
 - B ● All load modules to be loaded dynamically as independent units during startup. These
B are generated with LOAD-MODULE LOAD-MODE=STARTUP. The load modules are
B loaded in accordance with the sequence of LOAD-MODULE statements defined in this
B way.

B Load modules generated with LOAD-MODE=ONCALL are loaded the first time an assigned
B program unit is called.

B Please note the following:

B ● Load modules containing TCB entries **cannot** be exchanged.

B ● When dynamically linking a load module with the generation ALTERNATE-
B LIBRARIES=YES, you must ensure that only RTS modules are actually linked. This is
B because when load modules are exchanged, only the load module itself is removed
B from memory. If the load module is used to dynamically load other modules using the
B autolink function, these modules remain in memory following the exchange process
B even though shared data structures, for example, have been modified.

B ● When linking with the SYSLNK.CRTE.PARTIAL-BIND library, the entry ALTERNATE-
B LIBRARIES=YES is not required for load modules that only contain C code and possibly
B data objects (areas), and should therefore be avoided.

B LOAD-MODULE_ lmodname

B [,ALTERNATE-LIBRARIES={ NO | YES }

B [,LIB=libname]

B [,LOAD-MODE={ STARTUP |
B ONCALL |
B STATIC |
B (POOL,poolname,{ NO-PRIVATE-SLICE |
B STARTUP |
B ONCALL })
B }]

B ,VERSION=version

B lmodname Name of the load module up to 32 characters in length



B This name is subject to the same rules as the element names of a program
B library (see also [section "Format of names" on page 260](#)).

B **ALTERNATE-LIBRARIES=**

B This is used to control the autolink function when dynamically linking the
B private slice of the load module.

B NO The autolink function is not executed when linking the load module.

B Default: NO

B B B B B B B B	YES	The BLS autolink function is activated. For instance, if a load module requires other RTS modules for exchange purposes and these have not yet been loaded into memory, this function is used to load these RTS modules dynamically. Before starting the UTM application, the required RTS libraries must be reserved with Linkname BLSLIB nn ($00 \leq nn \leq 99$). If open external references cannot be resolved by the loaded modules, these libraries are then searched in ascending order (as specified in nn) for appropriate definitions when dynamically loading the load modules.
B B		ALTERNATE-LIBRARIES=YES may only be used to dynamically load RTS modules and not to dynamically load user programs.
B B		Further information on the Autolink function can be found in the openUTM manual "Using openUTM Applications under BS2000/OSD".
B B B		The operand values LOAD-MODE=STATIC / (POOL, $poolname$,NO-PRIVATE-SLICE) cannot be combined with ALTERNATE-LIBRARIES=YES. Such a combination will be rejected by the KDCDEF run.
B B	LIB=libname	Program library from which the load module is to be loaded dynamically. <i>libname</i> can be up to 54 characters in length.
B B B		If LOAD-MODE = STATIC, the LIB= operand is ignored. In all other cases, you must assign a value to LIB= either in the LOAD-MODULE statement or in a preceding DEFAULT statement.
B B	LOAD-MODE=	Load mode of the load module
B B B B B	STARTUP	The load module is loaded dynamically as an independent unit when the application is started. External references from the subsystem, from class 3/4 memory, and from all other modules of the UTM application which are already loaded are resolved. For runtime system functions, see also the description of the operand ALTERNATE-LIBRARIES=YES.
B B		Load modules which are generated with LOAD-MODE=STARTUP and which contain TCB entries must not be exchanged during runtime.
B		Default: STARTUP

B ONCALL The load module is loaded dynamically as an independent unit the first time
 B a program unit or conversation exit assigned to the load module is called.
 B External references from class 3/4 memory and from all other modules of
 B the UTM application which are already loaded are resolved.



B Load modules containing TCB entries must not be generated with
 B LOAD-MODE=ONCALL.

B If you are working with several processes, this load module must not be
 B overwritten in the library LIB=*libname* during runtime. Otherwise, different
 B versions of the load module will be executed during the application run.

B STATIC The load module must be linked in statically to the application program.

B (POOL,*poolname*, NO-PRIVATE-SLICE)

B The memory pool is defined using the MPOOL statement.

B *poolname* can be up to 50 characters in length.

B When the application is started, the load module is loaded into the common
 B memory pool *poolname*. It is not divided into public and private slices. A
 B private slice is therefore not linked (even statically) into the application
 B program.

B (POOL,*poolname*, STARTUP)

B When the application is started, the public slice of the load module is loaded
 B into the common memory pool *poolname*. The private slice belonging to the
 B load module is then loaded into the local task memory.

B (POOL,*poolname*, ONCALL)

B When the application is started, the public slice of the load module is loaded
 B into the common memory pool *poolname*. The private slice belonging to the
 B load module is then loaded into the local task memory when the first
 B program unit assigned to this load module is called.

B Only external references from class 3/4 memory, from the subsystems, and
 B from the local memory pool are resolved.

- B
B VERSION=version
Version number of the load module up to 24 characters in length.
- B
B This version number is subject to the same rules as the version numbers of elements of a program library.
- B
B
B
B If VERSION=@, then the BLS addresses the load module *lmodname* in a PLAM library, which was last entered in this PLAM library without an explicit version specification. If you work with explicit versions in LMS, you **cannot** use @ as the load module version.
- B
B
B The rules governing the versions of elements in a program library also apply to name allocation. However, there is one limitation: if *version* contains the character “.” then the version must start with a letter.

LPAP - define an LPAP partner for distributed processing based on LU6.1

The LPAP control statement allows you to define a logical access point for the partner application in the local application. An LPAP statement is only required if communication with the partner application is to be carried out using the LU6.1 protocol. This logical access point is known as an LPAP partner. For each LPAP partner, you must define a logical name, possibly administration authorization for the partner application, maximum values for the message queue of the LPAP partner, and logical properties for communication with the partner application based on the LU6.1 protocol.



For information about generating LU6.1 connections see [section “Distributed processing via the LU6.1 protocol” on page 76](#).

The CON statement is used to assign a real partner application to the LPAP partner (see the CON statement on [page 313](#)).

```
LPAP_      lpapname
           [ ,BUNDLE = master_lpap_name ]
           [ ,KSET=keysetname ]
           [ ,LNETNAME=local_netname ]
           [ ,PERMIT={ ADMIN | SATADM1 | ( ADMIN,SATADM )1 } ]
           [ ,QLEV=queue_level_number ]
           [ ,RNETNAME=remote_netname ]
           ,SESCHA=sescha_name
           [ ,STATUS={ ON | OFF } ]
```

B *additional operand under BS2000/OSD*

B [,NETPRIO={ MEDIUM | LOW }]

B ¹ only permitted under BS2000/OSD

lpapname LPAP partner name, i.e. the logical name of the partner application, which is used by the program units of the local application to address the partner application. *lpapname* applies only in the local application, and can be up to eight characters in length.

The specified name must be unique and must not be assigned to any other object in name class 1. See also [section “Uniqueness of names and addresses” on page 265](#).

Together with the LTERM names and the OSI-LPAP names, the LPAP names form a common name class.

BUNDLE=master_lpap_name

Name of the master LPAP.

If this operand is specified, the LPAP becomes a slave LPAP of an LU6.1-LPAP bundle.

You define the *master_lpap_name* with a MASTER-LU61-LPAP statement.

Messages sent to the master LPAP of an LPAP bundle with an APRO call are distributed to the slave LPAPs of this LPAP bundle by openUTM. This allows the application to distribute the messages to be sent across several partner applications of the same type without the need to program this explicitly.

KSET=keysetname

Name of the key set assigned to the partner application in the local application. The key set is defined using the KSET statement. The partner application can only start those services or address those remote services generated in the local application

- which are not locked, i.e. for which no lock code has been defined, and
- whose key codes are defined in the key set *keysetname*.

The local application can thus be secured against unauthorized access by the partner application.

Default: No key set,

i.e. only transaction codes that are not protected with lock codes can be started by the partner application.

LNETNAME=local_netname

This is required only for heterogeneous links. *local_netname* identifies the VTAM name defined for the UTM application in the CICS or IMS partner application.

Default: Blanks

B
B

NETPRIO=

Transport priority to be used on the transport connection assigned to this LPAP partner.

B

Default: MEDIUM

PERMIT=

Authorization level of the partner application

ADMIN

The partner application can execute administration functions in the local application.

- B** **SATADM** The partner application can execute preselection functions in the local application, i.e. it can activate and deactivate the SAT logging of certain events (UTM SAT administration authorization).
- B**
- B**
- B** **(ADMIN,SATADM)**
 The partner application can execute administration and preselection functions in the local application.
- Default:
 The partner application cannot execute administration functions in the local application.
- QLEV=queue_level_number**
 Maximum number of asynchronous messages that can be accommodated in the message queue of the LPAP partner. If this threshold value is exceeded, further APRO-AM calls to this LPAP partner are rejected with UTM message 40Z.
- Default: 32767
 Minimum value: 0
 Maximum value: 32767 (i.e. unlimited)
- RNETNAME=remote_netname**
 This parameter is required only for heterogeneous links. *remote_netname* identifies the VTAM name of the CICS or IMS partner application.
- Default: Blanks
- SESCHA=sescha_name**
 The session characteristics that apply for communication between the local application and the partner application are defined under *sescha_name* in the SESCHA statement (see [page 470](#)). By specifying *sescha_name* here, you can assign this set of session characteristics to the LPAP partner.
- This is a mandatory operand.
- STATUS=** Specifies whether the LPAP partner is locked. The status can be changed during operation using the administration command KDCLPAP.
- ON** The LPAP partner is not locked. Connections can be established between the partner application and the local application or connections already exist.
- Default: ON
- OFF** The LPAP partner is locked. No connections can be established between the partner application and the local application.

LSES - define a session name for distributed processing based on LU6.1

The LSES control statement required only for communication based on the LU6.1 protocol.



For more information about generating LU6.1 connection see [section “Distributed processing via the LU6.1 protocol” on page 76](#).

It allows you to define a common session name for the connection established between two applications for distributed processing. This name is then used to resume an interrupted communication process. LSES also enables you to allocate the session to an LPAP partner. For this purpose, each LPAP statement must be assigned at least one LSES statement. In the case of parallel sessions, several different session names must be defined for the LPAP partner *lpapname*.

An LPAP partner must always be assigned the same number of sessions (LSES statement) and transport connections (CON statement).

Exception: More LSES statements than CON statements can be assigned to an LPAP partner for a UTM cluster application.

If a session is defined for the local application with LSES AAA, RSES=BBB, this session must be defined with LSES BBB, RSES=AAA in the generation of the partner application.

To ensure that the USER and session name need not be unique in two connected applications, the common session name consists of two parts:

sessionname = local_sessionname + remote_sessionname

```
LSES_      local_sessionname
           ,LPAP=lpapname
           [ ,RSES=remote_sessionname ]
```

local_sessionname

Name of the session in the local application.

The specified name must be unique and must not be assigned to any other object in name class 2. See also [section “Uniqueness of names and addresses” on page 265](#).

LPAP=lpapname

Name of the LPAP partner assigned to the partner application.

local_sessionname is used for communicating with the partner application assigned to the LPAP partner *lpapname* in the local application.

RSES=remote_sessionname

Remote half session name

Default: *remote_sessionname=local_sessionname* is set, if RSES is not named.

LTAC - define a transaction code for the partner application

The LTAC control statement allows you to define a local transaction code for a service or remote service program in a partner application. LTAC statements can be generated for communication based on both the LU6.1 protocol and the OSI TP protocol.

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.

```

LTAC_      ltacname
           [ , { ACCESS-LIST=keysetname | LOCK=lockcode } ]
           [ , LPAP=lpapname ]
           [ , LTACUNIT=ltacunit ]
           [ , RTAC={ C'rtacname' |
                    rtacname |
                    recipient_TPSU_title [ , CODE={ STANDARD |
                                                    PRINTABLE-STRING |
                                                    T61-STRING |
                                                    INTEGER }
                    ]
           ]
           }
           ]
           [ , STATUS = { ON | OFF } ]
           [ , TYPE={ D | A } ]
           [ , WAITTIME=( time1,time2 ) ]

```

ltacname Name of a local transaction code defined for the remote service program

ACCESS-LIST=*keysetname*

ACCESS-LIST= is used to specify the access authorizations that the user of the local UTM application must have in order to be able to send a job to the remote program. Whether the job is actually carried out by the remote application will depend on the access authorizations that are defined there.

ACCESS-LIST may not be specified in conjunction with the LOCK=*lockcode* operand.

For *keysetname* you must enter the name of a key set. The key set must be defined using a KSET statement.

A user can only access the LTAC if the key set of the user (USER ...,KSET=) contains at least one of the key codes contained in the key set (access list) *keysetname* of the LTACs.

If you enter neither ACCESS-LIST=*keysetname* nor LOCK=*lockcode* the LTAC is not protected and any user of the local application is able to start the remote service program.

Default: no key set

LOCK=*lockcode*

May not be specified in conjunction with the ACCESS-LIST= operand.

LOCK= specifies the Lock code of the remote service program. A service secured with a lock code can only be addressed by a program unit if the program unit was started under a user ID (KCBENID) and from a client or a partner application (KCLOGTER) whose key set contains a key code that matches the lock code.

If you enter neither ACCESS-LIST=*keysetname* nor LOCK=*lockcode* the LTAC is not protected and any user of the local UTM application is able to start the remote service program.

Default: 0 (no lock code)

Maximum value: Value of MAX ...,KEYVALUE=*number*

LPAP=*lpapname*

This identifies the partner application to which the service program belongs. You must enter the name of the LPAP partner assigned to this partner application or the name of an LU6.1-LPAP bundle or an OSI-LPAP bundle.

If the LPAP= operand is not specified, the name of the partner application must be entered in the APRO function call (in the KCPA field).

LTACUNIT=ltacunit

Specifies the number of accounting units that are calculated for each call of this LTAC in the accounting phase of the UTM accounting. The accounting units are added to the accounting unit counter of the user ID that called the LTAC.

You may only specify integer values. This operand is only relevant if you are using the “UTM Accounting” function. Further information on the UTM Accounting can be found in the openUTM manual “Using openUTM Applications”.

Default value: 1

Minimum value: 0

Maximum value: 4095

RTAC=

Name of the transaction code for the remote service program in the partner application. *ltacname* is used in the local application to address a service program defined under this transaction code (*recipient TPSU-title*) in the partner application.

Default: *rtacname=ltacname*

C'*rtacname*'

rtacname

recipient_TPSU_title

The name of the transaction code for the remote service program in the partner application (*recipient_TPS_title*) can be specified in the form of a character string or a number. A character string can be entered in the format C'*rtacname*' or *rtacname*.

For *recipient_TPSU_title*, the OSI TP standard distinguishes between the code types printable string, T.61 string, and integer, which are used internally by openUTM to represent the RTAC name.

CODE=STANDARD

If *recipient_TPSU_title* is specified in the form of a character string, it can be up to eight characters in length. It can only contain characters that are permitted for TAC names. Further information can be found in [section “Format of names” on page 260](#).

CODE=STD must be used for communication based on the LU6.1 protocol, and is recommended if the partner application is a UTM application.

For communication based on the OSI TP protocol, CODE=PRINTABLE-STRING is used internally.

Default: STANDARD

CODE=PRINTABLE-STRING

The *recipient_TPSU_title* string can be up to 64 characters in length, and is case-sensitive.

If the partner application is a UTM application, *recipient_TPSU_title* can be up to eight characters in length. It can only contain characters permitted for TAC names. If these requirements are not met, the string can only be used for heterogeneous links based on the OSI TP protocol.

The following characters are permitted for the code type PRINTABLE-STRING:

- A, B, C, . . . , Z
- a, b, c, . . . , z
- 0, 1, 2, . . . , 9

and the following special characters:

apostrophe	'
hyphen	-
blank	␣
colon	:
question mark	?
equals sign	=
comma	,
plus sign	+
period	.
left parenthesis	(
right parenthesis)
slash	/

CODE=T61-STRING

With the code type T61-STRING, openUTM supports all characters of the code type PRINTABLE-STRING as well as the following special characters:

dollar sign	\$
greater than sign	>
less than sign	<
ampersand	&
commercial at	@
number sign	#
semicolon	;
percentage sign	%
asterisk	*
underscore	_

CODE=INTEGER

For *recipient_TPSU_title*, you can specify a positive integer between 0 and 67108863.

This is permitted only for partner applications which are not UTM applications and which communicate on the basis of the OSI TP protocol.

STATUS= This defines whether or not the *ltacname* of the remote service program is locked when the local application is started.

The value entered for STATUS= applies until it is changed using the KDCLTAC administration command.

ON The transaction code *ltacname* is not locked, i.e. jobs are accepted for the corresponding service program.

Default: ON

OFF The transaction code *ltacname* is locked, i.e. jobs are not accepted for the remote service program.

TYPE= This defines whether the remote service program is operated in dialog or asynchronous mode.

D The remote service program is operated in dialog mode.

Default: D

A The remote service program is operated in asynchronous mode.

WAITTIME=(time1,time2)

Maximum time spent waiting for a session to be reserved. By appropriately selecting this wait time, you can limit the wait time of a user on the terminal that requests the remote service.

time1

Number of seconds spent waiting for a session to be reserved (possibly including connection setup) or for an association to be established when starting a remote service program.

- *time1* ≠ 0 for asynchronous TACs:
An asynchronous job is always placed in the message queue of the partner application.
- *time1* ≠ 0 for dialog TACs:
A dialog job is accepted if a logical connection exists to the partner application.
- *time1* = 0 for asynchronous TACs:
An asynchronous job (FPUT job) that is not time-driven is only entered in the message queue of the partner application if there is a logical connection to the partner application. If there is no connection, then the FPUT call is rejected with the return code 40Z, KD13.
- *time1*=0 for dialog TACs:
If there is no session or association generated for which the local application is the contention winner, then the dialog job (APRO DM call) is rejected with 40Z, KD11. If there are sessions/associations for which the local application is the contention winner, but none are free when the program ends, then the transaction is rolled back.

In the case of asynchronous jobs to OSI TP partners *time1* is always set internally to 60 seconds, regardless of the value actually set.

If there is no logical connection to the partner application, then dialog jobs are rejected, regardless of the value of *time1*. The establishment of a connection is initiated at the same time.

time2 Maximum number of seconds spent waiting for a response from the job receiver. This can be used to restrict the wait time for the terminal user. *time2* = 0 means “wait indefinitely”.

time2 is only relevant for dialog LTACs, the wait times for asynchronous LTACs are defined using UTMD ... CONCTIME=(...,*time2*).



If a value > 0 is specified in *time2* then this value is ignored by openUTM if a KDCSHUT WARN or GRACE has been issued and the local service has initiated the end of the transaction. In this case, openUTM chooses the wait time in such a way that the transaction is rolled back before the application is terminated in order, if possible, to prevent the application from being terminated abnormally with ENDPET.

Default value: WAITTIME = (30,0).

Minimum value: WAITTIME = (0,0)

Maximum value: WAITTIME = (32767,32767)

Wait times can be modified using the UTM administration (e.g. with the KDCLTAC command).

LTERM - define an LTERM partner for a client or printer

The LTERM control statement allows you to define an LTERM partner as the logical access point for a client or printer of the application. Clients are terminals, UPIC clients and transport system applications (DCAM, PDN, CMX and socket applications, or UTM applications generated as transport system applications).

LTERM partners are used by clients and printers to establish a connection with the UTM application. They are assigned physical clients or printers using the PTERM statement. You can also define pools of LTERM partners; further information can be found in the description of the TPOOL statement on [page 511](#).

LTERM partners can also be predefined, i.e. they aren't assigned to a client/printer yet. The LTERM partner → PTERM assignment can be defined later on during operation using the KDCSWTCH administration command.

A separate LTERM statement must be issued for all clients defined in a PTERM statement.

W



Printers are not supported by openUTM under Windows systems.

```

LTERM_      ltermname
            [ ,BUNDLE=master-lterm]
            [ ,GROUP=primary-lterm]
            [ ,KSET=keysetname ]
            [ ,LOCK=lockcode ]
            [ ,QAMSG={ YES | NO } ]
            [ ,QLEV=queue_level_number ]
            [ ,RESTART={ YES | NO } ]
            [ ,STATUS={ ON | OFF } ]
            [ ,USAGE={ D | 0 } ]
            [ ,USER=username ]

```

B/X

BS2000/OSD and Unix system specific operands

B/X

```
[ ,CTERM=ltermname2 ]
```

B/X

```
[ ,PLEV=print_level_number ]
```

B

BS2000/OSD specific operands

B

```
[ ,ANNOAMSG={ Y | N } ]
```

B

```
[ ,FORMAT= { + | * | # }formatname ]
```

B

```
[ ,KERBEROS-DIALOG = { YES | NO } ]
```

B

```
[ ,LOCALE=( [ lang_id ][ , [ terr_id ][ ,ccsname ] ] ) ]
```

B

```
[ ,NETPRIO={ MEDIUM | LOW } ]
```



The operands LOCK=, KSET=, USER= and ANNOAMSG= are only valid for clients; the operands CTERM= and PLEV= are only valid for printers.

ltermname Name of the LTERM partner up to eight characters in length

ltermname is used

- to assign a client or printer to the LTERM partner in the PTERM statement.
- by the program units of the application to address clients, printers, and other TS applications (not server-to-server communication) assigned to the LTERM partner.

The specified name must be unique and must not be assigned to any other object in name class 1. See also [section “Uniqueness of names and addresses” on page 265](#).

B
B
B

ANNOAMSG= (**announce asynchronous message**)

This applies only for LTERM partners used by terminals (USAGE=D) to sign on to the UTM application.

B
B
B

Y

An asynchronous message to this terminal is announced in advance by outputting UTM message K012 in the system line. The user must then request the message using the UTM command KDCOUT.

B

Default: Y

B
B
B
B

N

An asynchronous message to this terminal is sent immediately, i.e. without prior announcement. If ANNOAMSG=N is generated, OMNIS V7.0 or later is required in order to establish a connection to this terminal via a multiplex connection (see the description of the MUX statement on [page 414](#)).

BUNDLE=master-lterm

Name of a master LTERM in a LTERM bundle (connection bundle). By specifying *master-lterm*, this LTERM becomes a slave LTERM of the corresponding connection bundle.

The master LTERM specified here must have been generated in a preceding LTERM statement. Do not assign a PTERM to a master LTERM.

Connection bundles permit load balancing (see the [section “LTERM bundle” on page 137](#)).

Connection bundles can be generated for APPLI or SOCKET connections (PTYPE operand of the corresponding PTERM statement).

BUNDLE must not be specified together with GROUP or CTERM.

B/X CTERM=ltermname2
 B/X (control terminal)
 B/X This only needs to be specified for LTERM partners generated for printers
 B/X (USAGE=O).
 B/X *ltermname2* is the name of an LTERM partner (up to eight characters in
 B/X length) which was configured as a printer control LTERM
 B/X (LTERM ...,USAGE=D). The printer control LTERM can be assigned one or
 B/X more LTERM partners which were configured for printers. It is used to
 B/X manage printers, print jobs, and printer queues.
 B/X Default: Blanks, i.e. no printer control LTERM

GROUP=primary-lterm

Name of a primary LTERM. By specifying *primary-lterm*, this LTERM becomes an alias LTERM of the corresponding primary LTERM. They define a LTERM group.

In a LTERM group you assign several LTERMs to one connection (see the [section "LTERM groups" on page 140](#)).

The primary LTERM specified here must have been generated in a preceding LTERM statement. The primary LTERM must be a normal LTERM assigned to a PTERM with PTYPE=APPLI or PTYPE=SOCKET or the master LTERM of a connection bundle. Do not assign a PTERM to an alias LTERM.



GROUP must not be specified together with BUNDLE or CTERM.

B FORMAT= Designates the start format of the LTERM partners. Start formats can only
 B be defined for terminals. It only makes sense to specify a start format if the
 B application is generated without user IDs or if you are using your own sign-
 B on service.
 B If the LTERM partner is assigned to a UPIC client, then specifying a start
 B format has no effect.
 B If the application is generated **without** user IDs, this format is output instead
 B of UTM message K001. Following a terminal-specific restart, the start
 B format is not displayed, rather the KDCDISP command is executed.
 B If the application is generated **with** user IDs, the name of the start format
 B can be queried in the first part of the sign-on procedure using the SIGN ST
 B call. If you do not use your own sign-on procedure, you cannot use the
 B LTERM-specific start format.

B		The sign of the format consists as follows:
B		+, * or # followed by an alphanumeric name (<i>formatname</i>) up to seven characters in length.
B		
B		#formats can only be used in the context of a sign-on procedure.
B		The terms have the following meanings:
B	+	When the next MGET call of the program unit is issued, each entry in a format field is preceded by 2 bytes for the attribute field in the KDCS message area, i.e. the field properties can be modified by the program unit. The format name at the KDCS interface is <i>+formatname</i> .
B		
B		
B		
B	*	When the next MGET call of the program unit is issued, the entry in a format field is not preceded by any bytes for an attribute field, i.e. the field properties cannot be modified by the program unit. The format name at the KDCS interface is <i>*formatname</i> .
B		
B		
B		
B	#	This identifies a format with extended user attributes. The field properties and global format properties can be modified by the program unit. The format name at the KDCS interface is <i>#formatname</i> .
B		
B		
B		Default: no start format
B	KERBEROS-DIALOG =	
B	YES	A Kerberos dialog is performed when a connection is established for terminals that support Kerberos and that connect to the application directly via this LTERM partner (not via OMNIS).
B		
B		
B		openUTM stores the Kerberos information in the length resulting from the maximum lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated to this length and stored.
B		The KDCS call INFO (KCOM=CD) allows a program unit run to read this information unless a user subsequently signs on with an ID card. In this event, the Kerberos information is overwritten by the ID card information.
B		If the maximum of the lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH is zero, a warning message is issued.
B	NO	No Kerberos dialog is performed.
B		
B		Default.

KSET=keysetname

This applies only to clients generated as dialog partners (USAGE=D). *keysetname* is the name of a key set defined using the KSET statement and assigned to the LTERM partner *ltermname*. *keysetname* may be up to 8 characters long.

A maximum of one key set can be assigned to each LTERM partner. This defines the access permissions for this LTERM partner with respect to using the services of the application and remote services (LTACs) generated in this application.

This LTERM partner can only be used to start services of the application that are protected with a lock code or an access list and only address remote services that are protected with a lock code or an access list if the following applies: The key set assigned to the LTERM partner and the KSET of the UTM user ID under which sign-on using this LTERM partner was performed must contain the key code or access code that matches the lock code or access list.

The lock/key code concept and the access list concept are described in detail in the openUTM manual "Concepts und Functions". An introduction to data access control can be found in [section "Lock/key code concept" on page 219](#).

Services whose TACs are not secured with codes can be called by the user or the client program without restriction.

In the case of an application in which user IDs have been defined and for which data access control is not required for terminals, you can assign all key codes to the terminals as follows:

```
LTERM_... ,KSET=MASTERSET  
KSET_MASTERSET ,KEYS=MASTER
```

Default: No key set

B	LOCALE=(lang_id,terr_id,ccs_name)	
B		Language environment of the client that signs on to the application via this LTERM partner.
B		
B	lang_id	Freely selectable language identifier for the client, up to two characters in length.
B		
B		The language identifier may be queried by the program units of the application, so that messages can be sent to the terminal in the communication partner's language.
B		
B	terr_id	Freely selectable territorial identifier for the client, up to two characters in length.
B		
B		The territorial identifier may be queried by the program units of the application, so that messages can be sent to the terminal taking into consideration any special territorial features of the communication partner's language.
B		
B	ccsname	(coded character set name)
B		Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). The character set must be compatible with an ISO character set supported by the terminal assigned to this LTERM partner.
B		
B		During generation, KDCDEF cannot check the validity of the CCS name under the BS2000 system or the compatibility condition. If you specify a CCS name which is not defined in XHCS, this results in a PEND ER when an attempt is made to establish a connection via this LTERM partner during runtime.
B		
B		The character set with the specified CCS name is used for:
B		
B		– outputting dialog messages on 8-bit terminals if the application is generated without user IDs or if a user is not signed on to the terminal, and another CCS name is not explicitly selected using an edit profile or a format.
B		
B		– outputting asynchronous messages on 8-bit terminals if another CCS name is not explicitly selected using an edit profile or a format.
B		
B		Default: Locale defined in the MAX statement

LOCK=lockcode

This applies only for clients (USAGE=D).

Lock code assigned to the LTERM partner as a logical numerical lock.

lockcode is a number between 1 and the maximum value permitted by the application (MAX ...,KEYVALUE=*number*). It is only possible to sign on to this LTERM partner under a UTM user ID (USER) for which a key set has been generated with a key code that contains the lock code of the LTERM partner.

If the application is generated without user IDs (no USER statement), the LOCK= operand is ignored.

Default: 0, i.e. no lock code

Maximum value: Value of MAX ...,KEYVALUE=*number*

B
B

NETPRIO=

Transport priority to be used on the transport connection assigned to this LTERM partner.

B
B

NETPRIO has no significance for LTERM partners that are assigned to a PTERM using PTYP=SOCKET or PTYP=*RSO.

B
B
B

Default:

MEDIUM for clients

LOW for printers

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

PLEV=print_level_number

(**print level**)

This applies only for printers. The PLEV= operand allows you to access printers from various UTM applications (printer sharing). The connection between the UTM application and the printer exists only while the print job is being transferred, thus allowing other applications to establish a connection as required.

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

This operand defines the number of printer messages at which openUTM attempts to establish a connection with the printer. openUTM continues to collect these messages until the threshold value defined with PLEV= is reached. It then establishes a logical connection to the printer. The connection is shut down again as soon as there are no further messages for this printer. Another application can then output messages to the printer if necessary. If the client is assigned a printer pool, openUTM attempts to establish a connection to all printers in the pool as soon as the threshold value is reached. When all messages have been sent, the connection to all printers in the pool is shut down again.

B/X
B/X
B/X

If the connection to the printer is shut down (e.g. by administration) even through the threshold value PLEV= is still exceeded, openUTM attempts to reestablish the connection at intervals defined in MAX ...,CONRTIME=*time*.

B/X
B/X

If PLEV=0 is specified, the connection is not shut down even if there are no further output messages.

B/X
B/X
B/X

If PLEV>0 is specified, the operands RESTART=NO or USAGE=D must not be specified for the LTERM partner. Under BS2000/OSD QAMSG=NO must also not be specified.

B/X
B/X

If PLEV>0 is specified, the operand CONNECT=YES of the associated PTERM statement has no effect.

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

Default value: 0

Minimum value: 0

Maximum value: 32767

If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.

QAMSG= (queue asynchronous message)

YES

An asynchronous message (FPUT job) to the client or printer is buffered by openUTM in the message queue of this LTERM partner, even if the client or printer is not connected to the application.

Default: If RESTART= YES

NO

An FPUT job sent to this client or printer is rejected with the return codes KCRCCC=44Z and KCRCDC=K705 if the client or printer is not connected to the application.

Default: If RESTART=NO

QLEV=queue_level_number

(queue level)

Specifies the maximum number of asynchronous messages simultaneously buffered by openUTM in the message queue of the LTERM partner. If this threshold value is exceeded, openUTM rejects all further FPUT or DPUT calls for this LTERM partner with 40Z.

QLEV= can be used to control the size of the page pool more effectively. This is because the number of asynchronous messages cannot exceed the *queue_level_number*.

Default value: 32767

Minimum value: 0

Maximum value: 32767 (i.e. unlimited)

If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.



You should not specify a QLEV < PLEV for a printer, as this would mean that the connection to this printer would have to be established by administration.

RESTART=	Processing of asynchronous messages when the client link is disconnected.
YES	When the link to the client assigned to this LTERM partner is disconnected, asynchronous messages are retained. If user IDs are not generated in this application, openUTM performs an automatic service restart for this LTERM partner. Default: YES
NO	When the link to the client assigned to this LTERM partner is disconnected, openUTM deletes all asynchronous messages in the message queue of the LTERM partner. If these are messages of a UTM message complex, the negative confirmation job is activated. It is possible to relieve the load on the page pool by specifying RESTART=NO. If QAMSG=YES is specified, you must not specify RESTART=NO. If user IDs are not defined in the application, openUTM does not perform an automatic service restart for clients or printers, i.e.: <ul style="list-style-type: none"> – If the connection is shut down by KDCCOFF, if it is lost, or if the application is terminated normally, the service is reset to the last synchronization point and terminated. The event exit VORGANG is then called with KCKNZVG=D (=Disconnect). – During a UTM warm start following abnormal termination of the application, an open service for this LTERM partner is terminated without calling the event exit VORGANG. – Following connection setup, KDCCDISP/KDCLAST behaves in the same way as after regeneration, i.e. the UTM message K020 NO MESSAGE(S) PRESENT is output.
STATUS=	Status of the LTERM partner following connection setup. This can be modified during runtime using the administration command KDCLTERM.
ON	The client or printer assigned to this LTERM partner is not locked, i.e. you can work with it as soon as the connection has been established. Default: ON
OFF	The client or printer assigned to this LTERM partner is locked.
USAGE=	Type of LTERM partner
D	The LTERM partner is configured as a dialog partner. Both the client and the application can send messages via the connection between the client and the local application. Default: D

- The LTERM partner is configured for an output medium. It is only possible to send messages from the application to the printer or TS application etc.

USER=username

This only applies if the LTERM has been configured as a dialog partner (USAGE=D). Depending on the type of the assigned client, this operand has the following effect:

- If a terminal is assigned to the LTERM partner, openUTM executes an automatic KDCSIGN for the user ID *username* when establishing a logical connection between the client assigned to this LTERM partner and the UTM application. Note that when the automatic KDCSIGN is used, access protection is limited. You should not specify this operand unless you are certain that an authorized user is working under this user ID at this client. After the logical connection is set up, the client is in the same state as if the user *username* executed the KDCSIGN command (BS2000/OSD) or sign-on check (Unix systems and Windows systems) successfully. The user ID must be defined using the USER statement.

- If the LTERM partner is assigned a client of type APPLI or UPIC, then the user ID *username* is reserved for this client (as the connection user ID). The client is signed on under this user ID when the connection is established. Another client or terminal user cannot sign on to the UTM application with this user ID.

If a user ID with the name of the LTERM partner was generated explicitly by means of a USER statement, openUTM assigns this user ID exclusively to the LTERM partner.

If the LTERM partner is to send administration calls to the application, then a user ID with administration authorization must be specified if the client is not signed-on using a real user ID.

If transaction codes are to be called from this client that are protected by lock codes, then this user ID must be assigned an appropriate key set.

If no user ID was specified, then KDCDEF implicitly creates a user ID with the name of the LTERM partner and the value defined in LTERM ...,RESTART=.

Default: No automatic KDCSIGN

MASTER-LU61-LPAP – Define the master LPAP of an LU6.1-LPAP bundle

The MASTER-LU61-LPAP statement allows you to specify the name and properties of a master LPAP for an LU6.1 LPAP bundle.

Slave LPAPs are assigned to a master LPAP of an LU6.1 LPAP bundle with the BUNDLE parameter of the LPAP statement. The master LPAP and the slave LPAPs together form an LPAP bundle. LPAP bundles allow messages to be distributed automatically across several LPAP partners (see the [section “LU6.1-LPAP bundles” on page 88](#)).

```
MASTER-LU61-LPAP  master_lpap_name
                  [ ,STATUS={ ON | OFF } ]
```

master_lpap_name

Name of the master LPAP of an LU6.1 LPAP bundle. This name is only of significance in the local application and must differ from the names of LTERMS, LPAPs, OSI-LPAPs and TACs defined in this application.

master_lpap_name can be up to 8 characters in length.

STATUS= Specifies whether the MASTER-LU61-LPAP is locked.

ON The MASTER-LU61-LPAP is not locked.

OFF The MASTER-LU61-LPAP is locked. Jobs for the MASTER-LU61-LPAP are rejected.

MASTER-OSI-LPAP - Defining the master LPAP of an OSI-LPAP bundle

With the control statement MASTER-OSI-LPAP you specify the name and properties of a master LPAP for an OSI-LPAP bundle.

A master LPAP of an OSI-LPAP bundle is assigned slave LPAPs with the BUNDLE parameter of the OSI-LPAP statement. The master LPAP and the slave LPAPs together form a LPAP bundle. LPAP bundles allow messages to be distributed automatically across several LPAP partners (see the [section “OSI-LPAP bundles” on page 105](#)).

```
MASTER-OSI-LPAP  master_lpap_name
                  ,APPLICATION-CONTEXT=context_name
                  [ ,STATUS={ ON | OFF } ]
```

master_lpap_name

Name for the master LPAP in an OSI-LPAP bundle. This name is only significant in the local application. It must be different from the names of the LTERMS, LPAPs, OSI-LPAPs, and TACs defined in the application.

master_lpap_name can be a maximum of 8 characters long.

APPLICATION-CONTEXT=context-name

Name of the application context to be used for communication with the remote partner.

All slave LPAPs in an OSI-LPAP bundle must be assigned to the same application context as the master LPAP.

STATUS= Specifies whether the MASTER-OSI-LPAP is locked.

ON The MASTER-OSI-LPAP is not locked.

OFF The MASTER-OSI-LPAP is locked. Jobs for the MASTER-OSI-LPAP are rejected.

MAX - define UTM application parameters

The MAX control statement allows you to define the maximum values, timers, process values and system parameters of a UTM application. For instance, these include:

- the name of the application
- the base name or the base directory for UTM files
- single or dual-file operation of the KDCFILE
- size of a UTM page (block size of UTM storages and buffers)
- the maximum number of
 - processes
 - key codes
 - GSSBs
 - LSSBs
 - UTM pages in the buffer for user log records, etc.
- threshold values for monitoring the size of SYSLOG file generations if the SYSLOG is created as an FG (SYSLOG-SIZE operand)
- B** ● the default language environment of the UTM application (LOCALE operand)
- B** ● whether or not SM2 can be used for performance monitoring in the application
- X** ● the network access mode, i.e. multi- or single-threaded.

The parameters of the MAX control statement can be split into several MAX statements. If the same operand is inadvertently entered in several MAX statements, the **first** value entered for this operand is taken as valid.

Mandatory operands:

APPLNAME=, KDCFILE= and TASKS=.

X/W *Additional mandatory operands under Unix systems and Windows systems:*

X/W SEMKEY= or SEMARRAY= (semaphore keys), IPCSHMKEY=, KAASHMKEY= and
X/W CACHESHMKEY=.

X/W In OSI TP applications additionally: XAPTPSHMKEY= and OSISHMKEY=.

The mandatory operands need to be defined once.

Note on UTM cluster applications:

If you modify one of the operands APPLINAME, APPLIMODE, GSSBS, LSSBS, KB, NB, VGMSIZE then you must regenerate both the initial KDCFILE and the UTM cluster files by specifying GEN=(CLUSTER,KDCFILE) in the OPTION statement.

For clarity, all operands of the MAX statement are listed in a table following the operand descriptions.

Operands valid for all operating systems

```

MAX_  [ APPLIMODE={ SECURE | FAST } ]
      [ ,APPLINAME=appliname
      [ ,ASYNTASKS={ atask_number | (atask_number,service_number) } ]
      [ ,BLKSIZE={ 2K | 4K } ]
      [ ,CACHESIZE=( number,paging,{ NORES | RES }1 ) ]
      [ ,CLRCH={ c | C'c'| X'xx' } ]
      [ ,CONN-USERS=number ] (mandatory under Unix systems and Windows systems)
      [ ,CONRTIME=time ]
      [ ,DEAD-LETTER-Q-ALARM=number ]
      [ ,DESTADM=destination ]
      [ ,DPUTLIMIT1=( day,hour,minute,second ) ]
      [ ,DPUTLIMIT2=( day,hour,minute,second ) ]
      [ ,GSSBS=number ]
      [ ,HOSTNAME=name ]
      [ ,KB=length ]
      [ ,KDCFILE=( filebase [, { SINGLE | DOUBLE } ] ) ]
      [ ,KEYVALUE=number ]
      [ ,LEADING-SPACES={ NO | YES } ]
      [ ,LPUTBUF=number ]
      [ ,LPUTLTH=length ]
      [ ,LSSBS=number ]
      [ ,NB=length ]
      [ ,NRCONV=number ]

```

B ¹ NORES | RES only permitted under BS2000/OSD

continued:

```
MAX_      [ ,OSI-SCRATCH-AREA=value ]
          [ ,PGPOOL=( number ,warnlevel1 ,warnlevel2 ) ]
          [ ,PGPOOLFS=number ]
          [ ,PGWTTIME=time ]
          [ ,QTIME = (qtime1,qtime2)]
          [ ,RECBUF=( number ,length ) ]
          [ ,RECBUFFS=number ]
          [ ,REDELIVERY=(number1, number2) ]
          [ ,RESWAIT={ time1 | ( time1, time2 ) } ]
          [ ,SM2={ NO | OFF | ON } ]
          [ ,SPAB=length ]
          [ ,STATISTICS-MSG={ NONE | FULL-HOUR } ]
          [ ,SYSLOG-SIZE=size ]
            ,TASKS=number
          [ ,TASKS-IN-PGWT=number ]
          [ ,TERMWAIT=time ]
          [ ,TRACEREC=number ]
          [ ,TRMSGLTH=length ]
          [ ,USLOG={ SINGLE | DOUBLE } ]
```

continued:

B	MAX_	<i>further operands for BS2000/OSD</i>
B		[,BRETRYNR=number]
B		[,CARDLTH=length]
B		[,CATID=(catalog_A,catalog_B)]
B		[,LOCALE=([lang_id] [, [terr_id] [,ccsname]])]
B		[,LOGACKWAIT=time]
B		[,MP-WAIT=number]
B		[,PRINCIPAL-LTH=length]
B		[,REQNR=number]
B		[,SAT={ ON <u>OFF</u> }]]
B		[,VGMSIZE=number]
X/W		<i>further operands for Unix systems and Windows systems</i>
X/W		,CACHESHMKEY=number
X/W		,IPCSHMKEY=number
X/W		[,IPCTRACE=number]
X/W		,KAASHMKEY=number
X/W		[,NET-ACCESS={ <u>MULTI-THREADED</u> SINGLE-THREADED }]
X/W		,OSISHMKEY=number <i>only mandatory if you generate OSI TP partners</i>
X/W		, { SEMARRAY=(number,number1) SEMKEY=(number,...) }
X/W		,XAPTPSHMKEY=number <i>only mandatory if you generate OSI TP partners</i>

APPLIMODE= This specifies whether the application is a UTM-S or UTM-F application.

SECURE The application is generated as a UTM-S application.

With UTM-S, openUTM logs all user data so that this data is retained after the application is terminated or following a system crash. In the event of errors, UTM-S guarantees the integrity and consistency of the application data. If a UTM-S application is terminated abnormally, an automatic restart is automatically performed. For this purpose, this variant of openUTM logs all modifications at the end of transactions.

Default: SECURE

FAST The application is generated as a UTM-F application.

UTM-F offers enhanced performance by eliminating the disk input/output operations performed by UTM-S when logging user and transaction data. With a standalone UTM-F application, openUTM only logs user passwords and changes to the configuration which were made by means of dynamic administration. These modifications are thus retained for the next application run. However, UTM-F applications do not log changes to the user data. They are therefore suitable only for installations in which performance is the most important criterion and the restart facility is not required. This applies in the case of pure information systems, or if all logging functions can be provided by the database system used.

In UTM cluster applications, user data that is valid globally in the cluster is also saved for UTM-F.

APPLINAME=appliname

Name of the UTM application up to eight characters in length

This is a mandatory operand.

If several application names are required, for example, for distributed processing based on the LU6.1 protocol, these can be assigned to the application using the BCAMAPPL statement. With APPLINAME= you define the primary application name.

appliname must be unique within the local system and may not begin with the character '\$'.

B

BS2000/OSD:

B

This name is subject to the name conventions for BCAM applications.

B

B

B

appliname must not begin with a number or with '\$' as this is prohibited by BCAM and the application cannot be started otherwise. Please note that KDCDEF cannot intercept numbers.

X/W

Unix systems and Windows systems:

X/W

X/W

appliname must be specified when establishing a connection from the terminal (dialog terminal process).

X/W

X/W

X/W

If connections are to be established with partner applications using the application name defined with APPLINAME=, you must also issue an appropriate BCAMAPPL statement (see [page 291](#)).

ASYNTASKS=(atask_number,service_number)

Maximum number of resources that may be reserved to process asynchronous jobs.

atask_number

Maximum number of processes (BS2000 tasks or work processes under Unix systems/Windows systems) of the application which can simultaneously handle jobs with asynchronous transaction codes. This operand allows you to prevent long-running asynchronous processes from affecting dialog operation.

If ASYNTASKS=0, asynchronous TAC classes cannot be generated.

Default: 1

Minimum value: 0

Maximum value: TASKS -1

service_number

Maximum number of asynchronous services that may be open at the same time.

You should set *service_number* to be larger than *atask_number* when one of the two following cases can arise:

- Process switch while processing an asynchronous service:
If an asynchronous service consists of several program units and if the transaction code of a follow-up program (follow-up TAC after PEND PA/PR or PEND SP) is located in a different TAC lass than the calling program unit or the priority control is generated for TAC classes (TAC-PRIORITIES statement), then a process switch can occur during processing. The asynchronous service is inactive at first and does not allocate a UTM process, although it remains open.
- Dialogs initiated by synchronous services with LU6.1 or OSI TP partner applications:
If a dialog is initiated with a partner application within an asynchronous service (with APRO DM) and it must wait for a response from the partner (via PEND KP or PEND RE), then the asynchronous service remains open until the response arrives (or until a timeout), but it does not allocate a UTM process.

If these cases arise in the application and the value of *service_number* is too small, then the asynchronous processing may be temporarily blocked because *service_number* of inactive services already exist. New asynchronous services cannot be started although no UTM processes are processing asynchronous services at this time.

Default: *atask_number*

Minimum value: *atask_number*
Maximum value: 32767

BLKSIZE= Size of a UTM page

Please note that, depending on the BLKSIZE specification, each user storage area occupies at least 2K or 4K in the page pool.

You can only specify BLKSIZE=4K for UTM cluster applications.

Default for standalone UTM applications: 2K

Default for UTM cluster applications: 4K

Possible values: 2K, 4K

Maximum value: 4K

B
B
B



Under BS2000/OSD you must specify BLKSIZE=4K
– if the KDCFILE and the USLOG file are created on NK4 disks, or
– if the KDCFILE is to be used as a Hiperfile (high-performance file).

X/W
X/W

Under 32-bit Unix systems and under Windows systems you must specify BLKSIZE=4K if you want to use more than 1976 keys (MAX KEY=).

X
X

On 64-bit Unix systems openUTM automatically sets the value of BLKSIZE to 4K.

X
X

A UTM page size of 4K is implicitly set for UTM cluster applications with 64-bit Unix systems for UTM cluster applications.

B
B
B
B
B

BRETRYNR=number


Number of attempts made by openUTM to transfer a message to the transport system (BCAM) if BCAM cannot accept the message immediately. If this number is exceeded, the connection to the dialog partner is shut down.


B
B
B
B
B
B
B
B
B
B

BRETRYNR is irrelevant for asynchronous messages output to a dialog partner with PTYPE=APPLI (PTERM statement). If such a message from the transport system is rejected due to a temporary bottleneck, then openUTM releases the process, but does not clear down the connection. After waiting for three seconds, openUTM makes up to three attempts to transfer the message to BCAM. If after the third attempt BCAM still cannot accept the message, then openUTM waits for 3 more seconds before it makes another three attempts to send the message to BCAM. If still unsuccessful, it waits another 3 seconds before making another three attempts, and so on.

B
B
B

Default: 10
Minimum value: 1
Maximum value: 32767 (theoretical value)

X/W X/W X/W X/W X/W	CACHESHMKEY=number	Authorization key for the shared memory segment containing the global buffer for file access. Keys are global parameters under the Unix systems and Windows systems. You cannot specify more than one key. You must enter a decimal number for <i>number</i> .
X/W	This is a mandatory operand.	
	CACHESIZE=(number,paging,NORES or RES) (NORES, RES only under BS2000/OSD)	This specifies the size and properties of the cache memory (further information can be found in the openUTM manual "Concepts und Functions"). The values entered here affect the performance of your UTM application.
	number	Number of UTM pages in the cache. The size of each UTM page is defined in the BLKSIZE= operand. The cache is used for accessing the page pool, i.e. all input and output operations involving LSSBs, GSSBs, TLSs, LPUT messages, FPUT messages, MPUT messages to clients, and some types of UTM administrative data. Data is not written to the KDCFILE until the cache becomes full or the transaction is terminated. KDCDEF rounds up this number to a multiple of 32. Default value: 1024 (≅ 2 or 4 MB, depending on the value of BLKSIZE=) Minimum value: 32 (≅ 64 or 128 KB, depending on the value of BLKSIZE=; 1MB in the upper address space of BS2000/OSD) Maximum value: Depends on the hardware and operating system, but not larger than 16777184.
B B B B		Under BS2000/OSD the cache is located in a common memory pool whose size is always a multiple of 1 MB. The BS2000 system automatically rounds the value specified in CACHESIZE. CACHESIZE should be requested in multiples of 1 MB so that address space is not wasted.
	paging	Percentage of cache pages to be written to the KDCFILE in a single batch in the event of a bottleneck, thereby freeing space in the cache. This must correspond to at least eight pages. The value specified here can be modified using the administration command KDCAPPL CACHE=%_utm_pages. Default value: 70(%) Minimum value: 0, i.e. eight pages are swapped out Maximum value: 100 (%)

B	NORES	The cache is created as non-resident.
B		Default: NORES
B	RES	The cache is created as resident.
B		Resident cache offers enhanced performance in productive mode, as the cache paging algorithm is designed specifically for use with this type of cache.
B		The number of resident pages used in the creation of a resident cache cannot be checked using the COREBIAS operand of the BS2000 command BIAS.
B		
B		
B	CARDLTH=length	
B		Length of the ID card information in bytes. If the ID card reader is used for sign-on, openUTM stores the ID card information in the length resulting from the maximum of the length specified here and the value generated for MAX PRINCIPAL-LTH. If the information on the ID card is longer, it is truncated and stored in this length.
B		The KDCS call INFO (KCOM=CD) enables a program to read this information.
B		CARDLTH must be big enough to ensure that the following applies for all USER statements with
B		USER ..., CARD = (<i>pos</i> , <i>string</i>):
B		$pos + \text{length}(\text{string}) - 1 \leq \text{CARDLTH}$.
B		Default: 0
B		Maximum value: 255
B		When a value > 255 is specified, 255 is assumed.
B		No warning message is output.
B	CATID=(<i>catalog_A</i> , <i>catalog_B</i>)	
B		Catalog IDs to which your KDCFILE is assigned.
B		If you work with CATIDs, enter the base name without the CATID in KDCFILE= <i>filebase</i> (see page 383).
B		In the case of single-file operation of the KDCFILE, specify the CATID to which the KDCFILE is to be assigned in <i>catalog_A</i> . In this case, <i>catalog_B</i> is not specified.
B		In the case of dual-file operation of the KDCFILE (see section "The KDCFILE" on page 45ff), you can assign files with the suffix A to CATID <i>catalog_A</i> and files with the suffix B to <i>catalog_B</i> . If you only specify a value for <i>catalog_A</i> , both files are assigned to this CATID.

B
B

In UTM cluster applications, only single-file operation of the KDCFILE is permitted.

CLRCH=

Character with which the KB program area and the standard primary working area are overwritten at the end of a dialog step. Possible entries are:

c
C'*c*'
X'*xx*'

Where *c* is an alphanumeric character and *x* a hexadecimal character.

Default:

The communication area and standard primary working area are not overwritten.

CONN-USERS=number

This operand is used to control the load on the application.

It defines the maximum number of users that can work simultaneously with the application. In the case of an application for which user IDs have not been generated, CONN-USERS= can be used to define the maximum number of clients that can sign simultaneously on to the application via LTERM partners.

- CONN-USERS < number of users/clients
This prevents all users/clients from working simultaneously with the application.
- CONN-USERS=0
The number of simultaneously active users/clients is unrestricted.
- CONN-USERS > number of users/clients
The application load is not controlled. CONN-USERS= is ignored.

User IDs and clients generated with administration authorization can sign on to the UTM application, even if the maximum number of simultaneously active user IDs has already been reached.

B

Default value in BS2000/OSD: 0 (i.e. no restriction)

Minimum value: 0

Maximum value: 500000

X/W
X/W
X/W

CONN-USERS is a mandatory operand in Unix systems and Windows systems. Please note that *number* cannot be set to a higher value than the number of concurrent user licenses obtained.

CONRTIME=time

(connection request time)

If the logical connection fails, this operand specifies the number of minutes after which openUTM attempts to cyclically reestablish the connection to the following partners:

- TS applications (PTYPE=APPLI or PTYPE=SOCKET) which openUTM generates with automatic connection setup (PTERM ...,CONNECT=A,) provided that the connection was not terminated by an administration command or due to the IDLETIME timer running down (see the PTERM statement on [page 437f](#)).
- OSI TP or LU6.1 partner applications which were generated with automatic connection setup, provided that the connection was not terminated by an administration command or because of the expiry of an IDLETIME timer.
- OSI TP partner to which the asynchronous messages were sent and with which no connection existed at the creation time of the messages.
- Printers to which openUTM establishes a connection as soon as the number of print jobs for this printer exceeds the generated threshold value (LTERM ...,PLEV>0). On disconnection, the number of print jobs must be greater than or equal to the threshold value if openUTM is to attempt to re-establish the connection.
If CONRTIME≠0, openUTM also attempts to re-establish the connection if this was previously explicitly disconnected using an administration command.
- Printers to which openUTM automatically establishes a connection (PTERM ...,CONNECT=YES), provided that the connection was not terminated by an administration command.
- Message distributor (MUX) to which openUTM automatically establishes a connection on start-up, provided that the connection was not terminated by an administration command.

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

If a connection to this partner is not established when the application is started or the administration command KDCPTERM or KDCLPAP is issued, openUTM attempts to reestablish the connection at intervals specified in CONRTIME=.

If CONRTIME=0, openUTM does not make any attempt to set up the connection. *Exception:* A wait time of 10 minutes is set for asynchronous messages to OSI TP partners.

Default: 10 min.
Maximum value: 32767 min.

DEAD-LETTER-Q-ALARM

Controls monitoring the number of messages in the dead letter queue.

The K134 message is output each time the threshold is reached. for this message the destination MSGTAC can be defined in order to automate handling of the dead letter queue.

Default: 0, monitoring is disabled.

Maximum value: 65535

DESTADM=destination

Destination to which openUTM sends the results of administration calls processed asynchronously. For *destination*, you can specify:

- an LTERM partner
Exception: UPIC-LTERM partners are not permitted!
- the TAC of an asynchronous program
- the TAC queue (type=Q).

Default: Blanks, i.e. no destination; the results are thus lost.

DPUTLIMIT1=(day,hour,minute,second)

Defines the latest possible execution time of a job. Can be specified in relative or absolute time:

time of execution < time of DPUT call + DPUTLIMIT1

The following applies for time specifications in DPUTLIMIT1:

day Maximum value: 364

Minimum value: 0

hour Maximum value: 23

Minimum value: 0

minute Maximum value: 59

Minimum value: 0

second Maximum value: 59

Minimum value: 0

Default value: DPUTLIMIT1 (360, 0, 0, 0) = 360 days

Default value: DPUTLIMIT2 (1, 0, 0, 0) = 1 day

Minimum value: (0, 0, 0, 0)

Maximum value: (364, 23, 59, 59)

The following must apply for the DPUTLIMIT1 and DPUTLIMIT2 operands:

$DPUTLIMIT1 + DPUTLIMIT2 \leq (364, 23, 59, 59) < 365 \text{ days}$

i.e. if you enter (364, 23, 59, 59) for DPUTLIMIT1, you must specify DPUTLIMIT2=(0, 0, 0, 0).

DPUTLIMIT2=(day,hour,minute,second)

The time specification for the DPUT call does not contain a number for the year. Furthermore, the desired execution time may already have passed if the DPUT call was delayed.

For this reason, you must decide whether the execution time of a job with an absolute time specification should be attributed to the past, current, or next year.

Since DPUTLIMIT1 + DPUTLIMIT2 must be < 1 year, only one of these three alternatives will be in the permissible open time period (call time - DPUTLIMIT2, call time + DPUTLIMIT1):

- If the only alternative allowed is before the call time, then the DPUT is handled as an FPUT and executed as soon as possible.
- If the only alternative allowed is after the call time, then the DPUT is saved and only converted to an FPUT and executed at the alternative time.
- If none of the three alternatives are in the permissible time period, then the DPUT is rejected.

DPUTLIMIT2 therefore allows you to backdate the specified execution time into the past for time-driven jobs with absolute time specifications. You cannot backdate jobs with relative time specifications.

DPUTLIMIT1 restricts the predating of jobs with absolute or relative time specifications into the future only.

Example 1

DPUTLIMIT1 = (300,0,0,0)

DPUTLIMIT2 = (010,0,0,0)

The DPUT call time is (005,0,0,0). The current and last years are not leap years.

- DPUTs with relative time (000,0,0,0) to (299,23,59,59) are accepted.
- DPUTs with absolute times (001,0,0,0) to (005,0,0,0) and (360,0,0,1) to (365,23,59,59) are handled as FPUT.
- DPUTs with absolute time (005,0,0,1) to (304,23,59,59) are handled as DPUT.
- DPUTs with absolute time (305,0,0,0) to (360,0,0,1) are rejected.

```

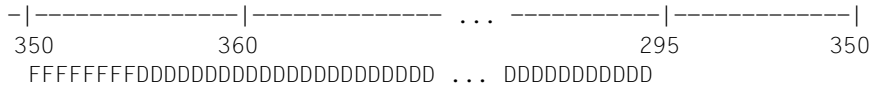
-|-----|----- . . . -----|-----|
360             5                               305           360
FFFFFFFFFFFFFFFFFDDDDDDDDDDDD . . . DDDDDDDDDDDDD

```

Example 2

DPUTLIMIT1 and DPUTLIMIT2 are defined exactly as in Example 1, but the DPUT call time is (360,0,0,0).

- DPUTs with relative time (000,0,0,0) to (299,23,59,59) are accepted.
- DPUTs with absolute time (350,0,0,1) to (360,0,0,0) are handled as FPUT.
- DPUTs with absolute time (001,0,0,0) to (294,23,59,59) and (360,0,0,1) to (365,23,59,59) are handled as DPUT.
- DPUTs with absolute time (295,0,0,0) to (350,0,0,0) are rejected.



The default values are listed under the description of the DPUTLIMIT1 operand.

GSSBS=number

Maximum number of GSSBs (global secondary storage areas) that can exist simultaneously in the application.

Default: 32
 Minimum value: 0
 Maximum value: 30000

HOSTNAME=name

Can only be specified in standalone applications.

In UTM cluster applications, you can specify a virtual host name in the VIRTUAL-HOST parameter of the CLUSTER-NODE statement.

- B
- B
- B
- B
- B
- X/W
- X/W
- X/W
- X/W
- X/W
- X/W

BS2000/OSD:
 Name of the virtual host on which the UTM application runs (from the point of view of BCAM). This virtual host must also be generated in BCAM.
 Default value:
 8 blanks, i.e the applications runs under the real host.

Unix systems and Windows systems:
 Name of the host that is specified as the sending address when a connection is established from the UTM application end. HOSTNAME= is required in cluster systems that use the “relocatable” IP address as the sending address and not the stationary IP address.
 Default: 8 blanks, the default processor name of the transport system is used as the sending address.

X/W	IPCSHMKEY=number	
X/W		Authorization key for the shared memory segment, which is used for communication between work processes on one side and the dialog terminal or printer processes and the timer process (external processes of an application) on the other side. Keys are global parameters under the Unix systems and Windows systems. You cannot specify more than one key. You must enter a decimal number for <i>number</i> .
X/W		
X/W		
X/W		
X/W		
X/W		
X/W		This is a mandatory operand.
X/W	IPCTRACE=number	
X/W		In test mode (startup with TESTMODE=ON, see openUTM manual “Using openUTM Applications under Unix systems and Windows systems”), openUTM writes entries in the trace area of the IPC (shared memory segments for interprocess communication). These entries contain internal information which is required for diagnostic purposes. Each entry occupies 32 bytes. IPCTRACE defines the number of entries in the trace area. If this number is exceeded, openUTM overwrites the existing entries, beginning with the oldest entry.
X/W		
X/W		Default: 1060
X/W		Minimum value: 1
X/W		Maximum value: 32500
X/W		
X/W		KDCDEF automatically resets values < 1 or > 32500 to the minimum or maximum value without outputting a UTM message.
X/W	KAASHMKEY=number	
X/W		Authorization key for the shared memory segment containing the global data. Keys are global parameters under the Unix systems and Windows systems. You cannot specify more than one key. You must enter a decimal number for <i>number</i> .
X/W		
X/W		
X/W		
X/W		This is a mandatory operand.
	KB=length	
		Length of the communication area (KB) in bytes, excluding the KB header and KB return area.
		Default: 512
		Minimum value: 0
		Maximum value: 32767

KDCFILE=

filebase Base name of the KDCFILE, the user log file, and the system log file SYSLOG. The name entered here must also be specified in the start parameter FILEBASE=*filebase* when starting the application program (see openUTM manual “Using openUTM Applications”).

This is a mandatory operand.

B

BS2000/OSD:

B

If you use the CATID= parameter to assign catalog IDs to your KDCFILE, the base name must be specified without a CATID. (see section “BS2000/OSD:” on page 46 for the format and length of the name).

B

B

X/W

Unix systems and Windows systems:

X/W

filebase is the name of the directory containing the KDCFILE and all application files. This directory must be created **before** the KDCDEF run.

X/W

filebase can be fully or partially qualified and can be a maximum of 29 characters in length for standalone applications, irrespective of whether the name is fully or partially qualified.

X/W

X/W

filebase can be a maximum of 27 characters in length for UTM cluster applications.

X/W

X/W

X/W

SINGLE Single-file operation is activated for the KDCFILE.

If the KDCFILE is split (see section “Splitting the KDCFILE” on page 57), all KDCFILE files are subject to single-file operation.

Default: SINGLE

DOUBLE For security reasons, dual-file operation is activated for the KDCFILE.

If the KDCFILE is split (see section “Splitting the KDCFILE” on page 57), all KDCFILE files are subject to dual-file operation.

In UTM cluster applications, only SINGLE may be specified.

KEYVALUE=number

Value of the highest key code of the application, and thus the value of the corresponding highest lock code that can be assigned to a transaction code or a terminal for data access control.

The operand `KEYVALUE=number` can also be used to define the maximum number of key codes per key set. `openUTM` uses this information to optimize the key set tables.

You can define up to 4000 key and lock codes. Only numerical lock codes can be defined.

Default: 32

Minimum value: 1

Maximum value (BS2000/OSD): 4000

Maximum value (32-bit Unix systems and Windows systems):

4000 for MAX ...,BLKSIZE=4K;

1976 for MAX ...,BLKSIZE=2K

Maximum value (64-bit Unix systems):

4000 for standalone UTM applications

3900 for UTM cluster applications

If you enter a value < 1, KDCDEF automatically sets `KEYVALUE=1` without outputting a UTM message.

B
X/W
X/W
X/W
X
X
X

LEADING-SPACES=

Specifies how the leading spaces in a messages from a terminal or from a TS application (`PTERM ... PTYPE=APPLI` or `SOCKET`) are to be handled.

YES When calling a program unit, leading blanks in messages are passed on to the program unit. The same applies for messages sent to a client with `PTYPE=APPLI`. A blank acting as a separator between TAC and message is removed if the TAC name < 8 characters.


Example

With the message, `TACNAME_ _Message` for a TAC called TACNAME in `FGET/MGET`, the following message will be sent to the program unit:

`_Message`

NO Leading blanks are suppressed.

Default: NO

B B B	LOCALE=(lang_id,terr_id,ccsname)	Default language environment of the UTM application (see also section "UTM messages" on page 186).
B B B B B	 The locale generated here is assigned to all user IDs and clients that sign on via LTERM partners or LTERM pools as the default language environment. This default setting applies unless another locale is explicitly defined for these objects in the corresponding USER, LTERM, or TPOOL statements.	
B B B B B B		The message module whose language and territorial identifiers match the specifications in the MESSAGE ...LOCALE= and MAX ...,LOCALE= statements becomes the application message module. openUTM sends messages to the message destinations SYSOUT, SYSLST, and CONSOLE from this application message module. The specifications in the application message module also determine the destination of a particular message.
B B	lang_id	Freely selectable language identifier for the UTM application up to two characters in length.
B		Default: Blanks
B B	terr_id	Freely selectable territorial identifier up to two characters in length.
B		Default: Blanks
B B B B B B	ccsname	(coded character set name) Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). During generation, openUTM cannot check whether this condition is fulfilled. KDCDEF will thus accept CCS names to which no character set is assigned.
B		Default: Blanks, i.e. 7-bit mode

B LOGACKWAIT=time

B The maximum length of time in seconds that openUTM is to wait for an
B acknowledgment from an output device. This acknowledgment is

- B** – for a printer, the logical print acknowledgment from the printer,
- B** – for an RSO printer, the acknowledgment from RSO,
- B** – for an FPUT call to another application, the transport acknowledgment.

B If confirmation does not arrive within this period, e.g. because the printer
B has run out of paper, openUTM shuts down the logical connection to the
B device.

B Default: 600

B Minimum value: 10

B Maximum value: 32767

LPUTBUF=number

Size of the LPUT buffer in UTM pages. The LPUT buffer of the KDCFILE is used to temporarily store LPUT data. This data is not copied to the user log file until the value specified in *number* is exceeded. The user log file USLOG is open only during this copy process.

Default: 1

Minimum value: 1

Maximum value: 1000

KDCDEF automatically resets values > 1000 to 1000 without outputting a UTM message.



CAUTION!

This operand must be set > 1 if the application contains LPUT calls. Otherwise, the copy process will be started too often. This involves opening and closing the user log files.

The value entered in LPUTBUF must be selected such that the buffer can accommodate the longest LPUT record. The following must apply:

$LPUTBUF * UTM \text{ page size} \geq LPUTLTH + \text{length of KB header (84 bytes)}$

LPUTLTH=length

Maximum length of the user data in LPUT records in bytes (excluding the KB header).

The maximum length of an LPUT record in the user log file is calculated as follows (see also the openUTM manual "Programming Applications with KDCS", user log file):

length + 84 bytes for the KB header + 12 bytes for length fields.

Default: 1948

Minimum value: 0

Maximum value (BS2000/OSD): 32652, irrespective of the storage medium for the user log file

Maximum value (Unix systems and Windows systems): 32668

B
B
X/W

BS2000/OSD:

B

openUTM uses *length* to determine the block size of the user log file. To do this, openUTM calculates the next largest value of (*length* + 100 bytes) that is a multiple of 2 kbytes. openUTM uses this multiple as the block length for the user log file. The 100 bytes comprise of 84 bytes for the KB header + 12 bytes for the record length fields + 4 bytes for the block length fields.

B
B
B
B
B

If the user log file USLOG is created on a non-key disk (NK2, NK4), then you must select the value of *length* such that:

B
B

length + 100 byte + 16 bytes block-specific internal DVS administration information

B
B

is a multiple of 2 Kbytes (on NK2 disks) or 4 Kbytes (on NK4 disks). This allows you to optimally utilize disk space.

B
B

The 16 byte block-specific internal DVS administration information are therefore not available for use as user data. You will find more information on this subject in the BS2000 manual "Introductory Guide to DMS".

B
B
B


LSSBS=number

Maximum number of LSSBs (local secondary storage areas) that can be created in a service.

Default: 8

Minimum value: 0

Maximum value: 1600

B B B	MP-WAIT=number	Maximum number of seconds for which openUTM waits for a process to sign on to a common memory pool.
B B B		Default value: 180 Minimum value: 1 Maximum value: 32000
B B B B		CAUTION! The default value of 180 seconds should only be changed in exceptional circumstances, e.g. if a process terminates with K078 ENQAR and a user dump with the return code KDCSST01.
	NB=length	Maximum length of a working area for <ul style="list-style-type: none"> – logical inputs and outputs to and from terminals and transport system applications of the APPLI type – asynchronous output messages to printers and transport system applications of the SOCKET type <p>This should be equal to the length of the largest KDCS message area of the program units in bytes. Under BS2000/OSD the value entered here must not exceed the value specified for TRMSGLTH. Under Unix systems and Windows systems the value of <i>length</i> must not exceed the value TRMSGLTH - 24 (see MAX ...,TRMSGLTH=).</p> <p>Default: 2048 Minimum value: 2048 Maximum value (BS2000/OSD): 32700 Maximum value (Unix systems and Windows systems): 32676</p> <p><i>BS2000/OSD:</i> If RSO printers are defined, the size of the RSO buffer (REMOTE-BUFFER-SIZE in the SPOOL parameter file) must be greater than or equal to <i>length</i>. See page 176.</p>
X/W X/W X/W	NET-ACCESS=	This specifies whether the application accesses the network in single- or multi-threaded mode.
X/W X/W		The value specified in NET-ACCESS does not affect connections via the socket interface (native TCP IP generated with T-PROT=SOCKET).

X/W
X/W**MULTI-THREADED**

Several network connections are managed in a single network process.

X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W

Network connections are allocated to network processes on the basis of listener IDs that are assigned to the application name (BCAMAPPL statement, see [page 291](#)) and access points (ACCESS-POINT statement, see [page 277](#)) of your application. All connections with the same listener ID are managed by the same network process. If you specify NET-ACCESS= MULTI-THREADED, you should also generate listener IDs. Otherwise, all access points and application names are implicitly assigned the listener ID 0 and all connections are managed by the same network process.

X/W

Default: MULTI-THREADED

X/W
X/W**SINGLE-THREADED**

Each network connection is managed in a separate network process.

W
W
W
W

This type of network connection is not supported by Windows systems. If you enter SINGLE-THREADED in a UTM application that is to be run under Windows systems, then KDCDEF replaces the value with MULTI-THREADED without informing the user.

NRCONV=number

(number of **conversations**)

Maximum number of services that can be simultaneously stacked by the user. NRCONV=0 means that services cannot be stacked.

The following limits are valid:

Number of user IDs + maximum number of services that can be placed on the stack (number of services = *number* * number of user IDs) ≤ 500000

If the limit value of 500000 is exceeded (by the values specified for NRCONV in the RESERVE statement, see [page 461](#), and by the number of USER statements, see [page 530](#)), then openUTM automatically creates fewer entries for stacking services. In this case, not all users will be able to place *number* services on the stack.

Default: 0

Minimum value: 0

Maximum value: 15

X/W OSISHMKEY=number
 X/W Authorization key for the shared memory segment, which is used by OSS
 X/W for communication based on OSI TP. You must enter a decimal number for
 X/W *number*.
 X/W This is a mandatory operand if the application communicates on the basis
 X/W of OSI TP.

OSI-SCRATCH-AREA=value
 Size in KB of an internal UTM working area for dynamic data storage when
 using the OSI TP protocol.

Default: 256
 Minimum value: 128
 Maximum value: 32767

B Under BS2000/OSD this working area is automatically extended during
 B runtime, if required.

X/W Under Unix systems and Windows systems the size of the internal working
 X/W area must not be modified during runtime. It is recommended that you select
 X/W the default value. However, if this proves to be insufficient during operation,
 X/W increase the value of OSI-SCRATCH-AREA and repeat the generation
 X/W procedure.

PGPOOL=(number,warnlevel1,warnlevel2)
 Size of the page pool in UTM pages and the warning levels for utilization of
 the page pool.

number Number of UTM pages to be used for the page pool in the KDCFILE (see
[page 49](#)). The size of each UTM page is defined in the BLKSIZE= operand.
 Default: 100
 Minimum value: 20
 Maximum value: 16777215 - (2 * number of PGPOOLFS)

If you enter a value less than 20, KDCDEF automatically sets PGPOOL=20
 without outputting a UTM message.

X/W Under Unix systems and Windows systems the value of PGPOOL is always
 X/W an even number. If you enter an uneven number, openUTM subtracts 1 from
 X/W your entry.

warnlevel1
 Numeric value (percentage) indicating the page pool utilization level at
 which the first warning (UTM message K041) is output.

Default: 80
 Minimum value: 1
 Maximum value: 99

warnlevel2

Numeric value (percentage) indicating the page pool utilization level at which the second warning is to be output. If *warnlevel2* is exceeded, all asynchronous jobs are rejected. In this case, the user receives a K message, and a corresponding return code is sent to a program unit.

Default: 95

Minimum value: *warnlevel1* + 1

Maximum value: 100

PGPOOLFS=number

Number of files between which the page pool is to be split. If PGPOOLFS = 0, the page pool is located in the main file (under BS2000/OSD in the file filebase.KDCA, under Unix systems and Windows systems in the file KDCA in the *filebase* file directory). In the case of dual-file operation (MAX ...,KDCFILE=(...,DOUBLE)), the value specified in *number* does not include the two file copies.

The file names are defined by KDCDEF.

Default: 0, i.e. the page pool is located in the main file

Note that BS2000 files cannot be larger than 32 Gbytes in size.

Maximum value (BS2000/OSD): 99 (and PGPOOL=*number* / 2)

Maximum value (Unix systems and Windows systems): 10

Minimum value:

In BS2000/OSD, an individual UTM file must not be larger than 32 Gbytes in size.

On Unix systems in 32-bit mode and in Windows, files up to 2 Gbyte in size are supported.

On Unix systems in 64-bit mode, openUTM can also use larger files as defined by the limits of the operating system and file system.

This results in the following minimum value depending on the size of a UTM page:

2 if BLKSIZE = 4K and PGPOOL *number* ≥ 8388608

0: The page pool is located in the main file.

PGWTTIME=time

Maximum number of seconds for which a program unit can wait for messages to arrive after a blocking call (e.g. PGWT call). During this period, a process of the UTM application is exclusively reserved for this program unit.

Default: ≈ *time* in TERMWAIT=*time*

Minimum value: 60

Maximum value: 32767

B
X/W

B
B
B
X/W
X/W
X
X

B PRINCIPAL-LTH=length
 B Maximum length of a Kerberos principal in bytes.
 B This parameter is only of significance if at least one user is generated with
 B USER ..., PRINCIPAL= or at least one LTERM or TPOOL is generated with
 B KERBEROS-DIALOG=YES. The length of the value specified with USER ...
 B PRINCIPAL= must not be larger than the value generated with MAX
 B PRINCIPAL-LTH=.

B When a Kerberos dialog is performed with a client, openUTM saves the
 B Kerberos information in the length resulting from the maximum of this length
 B and the length generated for MAX CARDLTH. If the Kerberos information is
 B longer, it is truncated to this length and stored.
 B The KDCS call INFO (KCOM=CD) allows the program unit run to read this
 B information if no user signs on to the same client with an ID card after the
 B Kerberos dialog. In this event, the Kerberos information is overwritten by the
 B ID card information.

B Default: 0
 B Minimum value: 0
 B Maximum value: 100

QTIME = (qtime1, qtime2)

Specifies the maximum permitted length of time that a service is to wait for the arrival of a message in a message queue. QTIME= refers to user-specific (USER queues), permanent (TAC queues) and temporary message queues.

It is possible to define individual maximum values for wait times in dialog or asynchronous services.

If a greater wait time value is specified in a program unit run than is generated in QTIME=, openUTM resets the wait time to the generated value.

qtime1 Maximum length of wait time for dialog services

qtime2 Maximum length of wait time for asynchronous services

Both times are specified in seconds.

Default: 32767 (seconds)

Maximum value: 32767 (seconds)

Minimum value: 0 (seconds)

RECBUF=(number,length)

Size of the transaction-oriented restart area. This area contains the data required for a restart following a transaction or system error. Further information on the restart area can be found on [section "Restart area" on page 52](#).

number Number of UTM pages per process to be used in the KDCFILE to store data for a restart following a system error. The size of each UTM page is defined in the BLKSIZE= operand. If this area is large, the application load is reduced but the restart process following a system error is slower. If this area is small, the application load is increased but the restart process following a system error is faster.

Default: 100 (per process)

Minimum value: 5 (per process)

Maximum value: 32767 (per process)

length Size in bytes of the buffer available to each application process for temporarily storing restart data. This data is required for a restart following a transaction or system error.

Default: 8192

Minimum value: 1024

Maximum value: 16777212 (16 MB)

RECBUFFS=number

Number of files between which the restart area is to be split. If RECBUFFS=0, the restart area is located in the main file of KDCFILE. In the case of dual-file operation (MAX ..., KDCFILE=(...,DOUBLE)), the value specified in *number* does not include the two file copies. The file names are defined by KDCDEF.

number must not be greater than the maximum number of processes defined in TASKS=. If this requirement is not fulfilled, the default value is used.

Default: 0

Maximum value (BS2000/OSD): 99, or value of the TASKS parameter

Maximum value (Unix systems and Windows systems):

10, or value of the TASKS parameter

B
X/W
X/W

REDELIVERY=(number1, number2)

Maximum number of redeliveries of an asynchronous message after the service or transaction was reset. *number1* and *number2* apply for different message destinations.

number1 Maximum number of redeliveries of messages to an asynchronous TAC. Delivery is always repeated after an asynchronous service was terminated abnormally with PEND ER/FR or system PEND ER without at least one transaction having been completed successfully. Restart of an asynchronous service after PEND RS within the first transaction is not regarded as a redelivery.

When redelivery is made, the program unit assigned to the TAC is restarted. With the FGET call, the number of redeliveries is output in the KB return area.

number2 Maximum number of redeliveries of messages to a service-controlled queue. Delivery is always repeated if the message was processed and the transaction was then reset.

With the DGET call, the number of redeliveries is output in the KB return area.

Default: (0, 255)

Minimum value for *number1* and *number2*: 0

Maximum value for *number1* and *number2*: 255 (i.e. the number is unlimited)

A value of 0 means that the message is deleted or saved to the dead letter queue after reset, depending on the value in TAC ...,DEAD-LETTER-Q.

If the value is set to 255, a message is redelivered any number of times. Note that this can result in an endless loop if, for example, a program unit is reset because of a programming error. Additionally the message cannot be saved to the dead letter queue in case of an endless loop.

B REQNR=number

B Maximum number of PAM read/write jobs that can be issued in parallel at the same time for a file in a UTM process. This value can be used to control the parallel processing of input/output operations within certain limits.

B Default: 20

B Minimum value: 1

B Maximum value: 100

B KDCDEF replaces an invalid value with the maximum value without outputting a message.

RESWAIT=(time1,time2)

(**resource wait**)

The times specified for *time1* and *time2* can be modified during runtime using the administration command KDCAPPL.

time1 Maximum number of seconds for which a program unit can wait for a resource locked by another transaction: GSSBs, TLSs, ULSs, and under BS2000/OSD possibly LTERM partners if ANNOAMSG=N.

If the resource does not become available within this time, the program unit receives an appropriate return code.

If the transaction currently occupying the resource is waiting for an input message following a PEND KP or PGWT KP program call, the program unit receives an appropriate return code immediately without having to wait for the period specified in *time1*. If a PEND KP or PGWT KP call is issued in a blocking transaction, all pending program units are informed of this by means of a return code.

RESWAIT=0: The application program does not wait for the resource to become available. If the resource is locked by another transaction, the requesting program unit immediately receives an appropriate return code.

Default: 120

Minimum value: 0

Maximum value: 32767

B
B



Under BS2000/OSD the real waiting time depends on the precision with which the bourse waiting time was set in the operating system.

time2

Maximum number of seconds for which you can wait for a resource locked by another process. If *time2* is exceeded, the application is terminated abnormally.

time2 should not be set too low, since certain activities in the UTM application must be performed and completed by a process before the same activities can be initiated in another process.

Example

When sending a message, a process locks the terminal to which the message is directed. If another process wishes to access an input message of the same terminal, it must wait for the terminal to become available again.

In particular, the value entered for *time2* must be at least equal to the longest processing time (real time) required in the following cases:

- In the case of a communication partner generated with PTERM ...,PTYPE=APPLI, the resources are locked for the entire duration of a processing step. This includes the time required to process the event exit VORGANG at the start and/or end of a conversation.
- At the end of a conversation, the resources remain locked as long as the event exit VORGANG is running.


Default: 300

Minimum value: 300

Maximum value: 32767

If the value 0 is specified for *time2*, KDCDEF uses the default value 300 without outputting a UTM message. If you specify a value between 0 and 300, however, KDCDEF issues an appropriate UTM message.

B B B B	SAT=	(security audit trail) Minimum event logging with SAT. Further information about "SAT logging" can be found in the openUTM manual "Using openUTM Applications under BS2000/OSD".
B	ON	SAT logging is switched on. Minimum logging with SAT is switched on for the following events: <ul style="list-style-type: none"> – signing a process on to and off from the UTM application – switching the memory protection key – exchanging programs – executing a UTM SAT administration command.
B B B B		Minimum logging can be extended and controlled by means of preselection. This is generated using the SATSEL statement and the SATSEL= operand in the USER and TAC statements. The administration command KDCMSAT can be used to modify the preselection values defined during generation.
B B B B B B	OFF	SAT logging is switched off The logging procedure only covers attempts to access the SAT administration TAC KDCMSAT (apart from KDCMSAT HELP). All other events are ignored. SAT logging can be switched on and off using the SAT administration TAC KDCMSAT (see the openUTM manual "Using openUTM Applications under BS2000/OSD").
B		Default: OFF
X/W X/W X/W X/W X/W X/W X/W	SEMARRAY=(number,number1)	Range of semaphore keys for global semaphores (process synchronization). Semaphore keys are global parameters under the Unix systems and Windows systems. With SEMARRAY, you enter an initial value <i>number</i> and an upper limit <i>number</i> . openUTM then reserves these keys, incrementing them by 1 starting with the initial value. For further information, please contact your system administrator.
X/W X/W X/W X/W X/W X/W X/W	<div style="border: 1px solid black; padding: 2px; display: inline-block; text-align: center; width: 20px; height: 20px; line-height: 20px;">i</div>	The SEMARRAY= and SEMKEY= parameters are mutually exclusive. Compared to SEMKEY=, SEMARRAY= offers the advantage of allowing openUTM to reserve more than ten semaphore keys. To calculate the number of semaphore keys required for a UTM application, see the following description of SEMKEY and the description of the global system resources in the openUTM manual "Using openUTM Applications under Unix systems and Windows systems".
X/W		This is a mandatory operand if SEMKEY= is not specified.
X/W	number	Initial value (numeric value)

X/W	number1	Number of keys to be reserved
X/W		Minimum value: 1
X/W		Maximum value: 1000
X/W	SEMKEY=(number,...)	
X/W		(semaphore key)
X/W		Semaphore keys for global semaphores (process synchronization).
X/W		Semaphore keys are global parameters under the Unix systems and
X/W		Windows systems. You can define up to 10 semaphore keys in a list. All
X/W		semaphore keys (<i>number...</i>) are specified in the form of a decimal number.
X/W		For further information, please contact your system administrator.
X/W		The SEMARRAY= and SEMKEY= parameters are mutually exclusive.
X/W		This is a mandatory operand if SEMARRAY is not specified.
X/W		The following formula can be used to calculate the number of semaphore
X/W		keys required for a UTM application. The resulting value should be rounded
X/W		up to the nearest integer.
X/W		number of semaphore keys = $(9 + n + m) / 10$
X/W		Where:
X/W		– 9 = constant value required by openUTM for internal synchronization
X/W		– n = maximum number of work processes that can be simultaneously
X/W		active during runtime
X/W		– m = maximum number of external processes that can be simultaneously
X/W		connected during runtime
X/W		For more information on this subject see also: openUTM manual “Using
X/W		openUTM Applications under Unix systems and Windows systems”, Global
X/W		system resources.
	SM2=	This defines whether the UTM application is to supply data to SM2 or
		openSM2 for performance monitoring.
	NO	Performance monitoring with SM2/openSM2 is generally prohibited for the
		UTM application, i.e. the UTM application cannot supply data to
		SM2/openSM2, nor can this be explicitly activated by the UTM adminis-
		trator.
	OFF	The UTM application can supply data to SM2/openSM2, but this must be
		explicitly activated by the administrator using KDCAPPL SM2=ON. The
		supply of data can be deactivated again at any time using the administration
		command KDCAPPL=OFF.
		Default value: OFF

- ON The UTM application can supply data to SM2/openSM2. This is activated automatically when starting the UTM application. It can be deactivated again at any time by the UTM application administrator using the administration command `KDCAPPL SM2=OFF`.
- SPAB=length Maximum length of the standard primary working area in bytes
 Default: 512
 Minimum value: 0
 Maximum value: 32767
- STATISTICS-MSG=
 Specifies whether or not openUTM is to produce statistics message K081 hourly.
- FULL-HOUR
 Statistics message K081 is produced every hour and written in the SYSLOG. At the same time, openUTM resets the following application-specific statistical values to 0:
- number of messages received (*term_input_msgs*)
 - number of messages sent/output (*term_output_msgs*)
 - number of requests to write records in the user log file USLOG (*logfile_writes*)
 - percentage of requests from buffers in the cache that led to wait times (*cache_wait_buffer*)
- NONE Statistics message K081 is not produced and the statistical values listed above are not automatically reset to 0.
 You should choose NONE if you want to reset the statistical values listed above via the administration when needed (see the openUTM manual “Administering Applications”, `KC_MODIFY_OBJECT`).
- Default: FULL-HOUR
- SYSLOG-SIZE=size
 Automatic size monitoring of the system log file SYSLOG by openUTM.
- *size*≠0
 This can only be specified if the system log file SYSLOG is created as a file generation group (FGG). If SYSLOG is a normal file and a value other than 0 is entered for *size*, openUTM aborts the application startup with the start error 58. If SYSLOG is created as an FGG, you can use SYSLOG-SIZE to activate the automatic size monitoring of the SYSLOG by openUTM. In this case, *size* defines the file generation size at which openUTM switches to the next file generation.

- *size=0*
If the value 0 is specified for *size* (default), openUTM does not monitor the size of the SYSLOG file. Instead, it outputs all UTM messages directed to SYSLOG to the same file generation until openUTM switches to another file generation by means of administration (KDCSLOG command), or until size monitoring is activated.
- *size≥100*
Values ≥ 100 are interpreted by openUTM as follows: the size of each individual SYSLOG file generation must not exceed the value (*size* * size of a UTM page). The size of each UTM page is defined in BLKSIZE. When the size of the SYSLOG file exceeds this threshold value, openUTM automatically switches to the next SYSLOG file generation.
- *size<100*
openUTM automatically resets values between 1 and 99 to 100. In this case, a UTM message is output for information purposes.
- *size<0*
Values < 0 are rejected by KDCDEF.

The administrator can modify the generated threshold value, and activate or deactivate size monitoring as desired during operation (e.g. with the KDCSLOG command).

Default: 0 (no size monitoring)

Minimum value: 100

Maximum value: ($2^{31} - 1$)

TASKS=number

Maximum number of processes that can be used simultaneously for the application.

This is a mandatory operand.

Minimum value: 2

Maximum value: 250

KDCDEF automatically resets values < 2 to 2 without outputting a UTM message.

The current number of processes is defined when starting the application. You can specify TASKS=1 during startup. The administrator can dynamically modify the number of processes during runtime (e.g. with the administration command KDCAPPL).

The number of processes specified during startup or set by the administrator must not exceed the value generated here.

TASKS-IN-PGWT=number

Maximum number of processes of the UTM application in which program units with blocking calls, e.g. the KDCS call PGWT, may run simultaneously. The value of TASKS-IN-PGWT must be less than that of the TASKS= operand.

If TASKS-IN-PGWT=0, it is not possible to generate a TAC class or a transaction code (TAC) for which blocking calls are permitted (see TAC/TACCLASS ...,PGWT=). In this case, PGWT=NO must be specified in all TACCLASS and TAC statement (see also the TAC statement on [page 483](#) and the TACCLASS statement on [page 500](#) for more information).

Default value: 0

Minimum value: 0

Maximum value: *number* in TASKS -1

TERMWAIT=time

(terminal wait)

time Maximum time in seconds that may elapse in a multi-step transaction (i.e. after PEND KP) between dialog output to the partner and the subsequent dialog response from the partner. This value applies for all dialogs in which the partner assumes the client role (terminals, UPIC clients, OSI TP, LU6.1 and LU6.2 job submitters). For terminal clients, for example, *time* is the time the user has to think after PEND KP.

In the event of a timeout, the transaction is reset and the resources reserved by the transaction are released. The connection to the partner is shut down.

Default: 600

Maximum value: 32767

Minimum value: 60

TRACEREC=number

Maximum number of entries in the process-specific trace areas handled by openUTM. This value applies to the trace area

- of the main routine KDCROOT (UTM-DIAGAREA)
- of the UTM system code (KTA trace)
- of the XAPTP module (XAP trace) for OSI TP applications

openUTM writes trace information to these areas for diagnostic purposes.

Length of the entries:

- Entry in UTM-DIAGAREA: 138 bytes (on 32-bit systems) or 256 bytes (on 64-bit systems)
- KTA and XAP trace entry: 64 bytes (on 32-bit systems) or 112 bytes (on 64-bit systems)

Default: 32500

Minimum value: 1

Maximum value: 32500 (depending on the available resources)

KDCDEF automatically resets values < 1 to the default value and values > 32500 to the maximum value without outputting a UTM message.

TRMSGLTH=length

This defines the maximum value for the following:

- The length of physical output messages sent to a terminal, printer or transport system application (PTYPE=APPLI) or received by a terminal or transport system application with PTYPE=APPLI. When the message length is calculated, all characters to be transmitted, including control characters etc., must be included.
- The length of asynchronous output messages to transport system applications of the SOCKET type.
- The length of the message section of the input message received from an UPIC client that uses TCP/IP via the socket interface. During the calculation of the length, it is necessary to take account of all the characters that are to be transferred, including protocol elements.

Default: 4096 bytes

Minimum value: 4096 bytes

Maximum value: 32700 bytes

B
B

In BS2000/OSD systems, the value specified in *length* must be at least as large as the value specified in the NB operand.

X/W
X/W

Under Unix systems and Windows systems the value entered in *length* must be at least 24 bytes greater than that specified for the NB= operand.

B
B
B
B

If you use RSO printers, the size of the RSO buffer (REMOTE-BUFFER-SIZE in the SPOOL parameter file) must be greater than or equal to TRMSGLTH=length. See also section [“Defining the RSO buffer size” on page 176](#) for more information.

USLOG= This defines single- or dual-file operation for the user log file USLOG.

SINGLE Single-file operation is activated for the user log file.

Default: SINGLE

DOUBLE For security reasons, dual-file operation is activated for the user log file. Further information on the user log file can be found in openUTM manual [“Using openUTM Applications”](#).

- VGMSIZE=number
This parameter is used to generate a buffer area with the specified size for the service memory of an SQL database system. It also restricts the user's share of the page pool. VGMSIZE= is specified in KB.
- If the service memory area to be logged when the PEND call is issued is greater than *number*, the service is terminated with PEND ER.
- Default value: 32KB

Minimum value: 32KB

Maximum value: 256KB
- XAPTPSHMKEY=number
Authorization key for the XAPTP shared memory segment
- Keys are global parameters under the Unix systems and Windows systems.
- XAPTPSHMKEY is a mandatory operand if the application is to communicate via the OSI TP protocol.

The table below provides an overview of the purpose and default values of the individual operands of the MAX statement:

	Operand	Purpose	Mandatory	Default value
	<i>Operands valid for all operating systems</i>			
	APPLIMODE=	Choice of UTM variant: UTM-S or UTM-F		SECURE
	APPLINAME=	Name of the UTM application	X	-
	ASYN TASKS=	Asynchronous processing (number of processes for asynchronous processing and asynchronous services open at the same time)		1, 1
	BLKSIZE=	Size of a UTM page		2K
 	CACHESIZE=	Tuning feature (size and properties of the cache)		BS2000/OSD: (1024,70%, NORES) Unix systems, Windows systems (1024,70%)
	CLRCH=	Character for overwriting the communication area and standard primary working area		None
 	CONN-USERS=	Restriction on the number of users or clients active simultaneously		BS2000/OSD: No restriction Unix systems, Windows systems: Mandatory operand

Operand	Purpose	Mandatory	Default value
CONRTIME=	Automatic connection setup for printers (waiting time for reconnection)		10 minutes
DEAD-LETTER-Q-ALARM=	Monitors the number of messages received in the dead letter queue		0, i.e. no monitoring
DESTADM=	Asynchronous administration		None
DPUTLIMIT1=	Time-driven jobs (upper limit)		360 days
DPUTLIMIT2=	Time-driven jobs (lower limit)		1 day
GSSBS=	GSSB storage areas (maximum number)		32
HOSTNAME=	Virtual host name for the UTM application		8 blanks
KB=	Maximum length of the communication area		512
KDCFILE=	Assigning a KDCFILE	X	-
KEYVALUE=	Data access control using the lock/key code concept (number of the highest key code)		32
LEADING-SPACES=	Pass leading blanks in messages from terminals or from TS applications (PTYPE=APPLI/SOCKET) to the program unit		NO
LPUTBUF=	Logging of user data with LPUT (number of PAM pages in the page pool)		1
LPUTLTH=	Logging of user data with LPUT (maximum LPUT message length)		1948 bytes
LSSBS=	LSSB storage areas (maximum number)		8
NB=	Maximum length of the KDCS message area		2048
NRCONV=	Maximum number of stacked services		0
OSI-SCRATCH-AREA=	Size in KB of an internal UTM working area		256
PGPOOL=	Size of the page pool and warning levels		100 UTM pages, 80%, 95%
PGPOOLFS=	Tuning feature: splitting the page pool		Page pool in KDCFILE
PGWTTIME=	Maximum time for the KDCS call PGWT		TERMWAIT= <i>time</i>
QTIME	Maximum permitted wait time for messages from service-controlled queues		32767 seconds
RECBUF=	Tuning feature: size of the restart area in KDCFILE or process-oriented system memory		5 PAM pages per process, 512 bytes

Operand	Purpose	Mandatory	Default value
RECBUFFS=	Tuning feature: splitting the restart area		In KDCFILE
REDELIVERY=	Maximum number of redeliveries of an asynchronous message		0 for UTM-controlled queues, 255 for service-controlled queues
RESWAIT=	Waiting time for a resource (e.g. GSSB, TLS) locked by another transaction (<i>time1</i>) or process (<i>time2</i>)		120 seconds 300 seconds
SPAB=	Maximum SPAB length		512
STATISTICS-MSG=	Statistics message K081 is produced and the counter is automatically reset to 0		FULL-HOUR
SYSLOG-SIZE=	Automatic size monitoring of the SYSLOG file by openUTM		0
TASKS=	Number of UTM processes	X	-
TASKS-IN-PGWT=	Number of processes for PGWT jobs		0
TERMWAIT=	Maximum waiting time for dialog input within a transaction		600 seconds
TRACEREC=	Space reserved for diagnostic information (number of entries)		32500
TRMSGLTH=	Maximum message length		4096 bytes
USLOG=	Single- or dual-file operation of the user log file		SINGLE
VGMSIZE=	Generate the buffer area with the specified size		32 KB
<i>BS2000/OSD-specific operands</i>			
B B	BRETRYNR=	Communication with BCAM (number of retries when sending messages)	10,0
B	CARDLTH=	ID card reader for KDCSIGN check	0
B	CATID=	Catalog IDs for the KDCFILE	Default CATID
B	LOCALE=	Default language environment	Blanks
B B	LOGACKWAIT=	Support for output devices (waiting time for confirmation)	600 seconds
B B	MP-WAIT=	Maximum waiting time per process for connection to the common memory pool	180 seconds

	Operand	Purpose	Mandatory	Default value
B B	PRINCIPAL-LTH=	Maximum length of a Kerberos principal in Byte		32
B B	REQNR=	Tuning feature: PAM I/O jobs (maximum number of parallel jobs)		2
B	SAT=	Minimum logging of events with SAT		OFF
B B	SM2=	Permitting, activating, and deactivating the supply of UTM data to SM2		OFF
B B	VGMSIZE=	Size of the buffer area for the service memory of an SQL database system		32KB
X/W	<i>Unix system- and Windows system-specific operands</i>			
X/W X/W	CACHESHMKEY=	Authorization key for a shared memory segment (global buffer for file access)	X	-
X/W X/W X/W	IPCSHMKEY=	Authorization key for a shared memory segment (communication between UTM processes)	X	-
X/W	IPCTRACE=	Number of UTM entries in the IPC trace area		1060
X/W X/W	KAASHMKEY=	Authorization key for a shared memory segment (global data)	X	-
X/W X/W	NET-ACCESS=	Mapping of network connections to network processes		MULTI-THREADED
X/W X/W	OSISHMKEY=	Authorization key for an OSS shared memory segment	With OSI TP	-
X/W X/W	SEMARRAY=	Range of semaphore keys for global semaphores (alternative to SEMKEY)	X	-
X/W X/W	SEMKEY=	Semaphore keys for global semaphores (alternative to SEMARRAY)	X	-
X/W X/W	XAPTPSHMKEY=	Authorization key for the XAPTP shared memory segment	With OSI TP	-

MESSAGE - define a UTM message module

The MESSAGE control statement allows you to incorporate user message modules in the configuration. It is possible to use a separate user message module to adapt the message texts and/or the message destinations of individual messages to suit your requirements.

For more information on message modules see also [section “UTM messages” on page 186](#) in this manual and the openUTM manual “Messages, Debugging and Diagnostics”.

B Generating message modules under BS2000/OSD

B In order to internationalize the application, it is possible to create several user message modules which output the UTM messages of an application in the appropriate language.

B The respective language environment can be defined for a user message module by means of a locale, i.e. a unique pair of language and territorial identifiers. The language-specific message modules are assigned for message output in accordance with the locale defined for the user and LTERM partner.

B The German UTM message module KCSMSGs and the standard English UTM message module KCSMSGSE are supplied with openUTM.


B MESSAGE_ MODULE=name
B [,LIB=onlname]
B [,LOCALE = (lang-id [,terr-id])]

B MODULE=name

B Name of the user-specific message module up to eight characters in length. This module is created using the KDCMMOD tool (see the openUTM manual “Messages, Debugging and Diagnostics under BS2000/OSD”).

B This is a mandatory operand.

B The name specified here must be unique within the application.

B B	LIB= <i>omlname</i>	Object module library from which the user-specific message module is to be loaded dynamically. <i>omlname</i> can be up to 54 characters in length.
B B		If the user-specific message module is to be loaded dynamically, it must not be linked to the application.
B B B B		If nothing is specified for LIB= , TASKLIB is assumed. This does not correspond to the SET-TASKLIB command, rather a library named TASKLIB must exist in this case. Dynamic loading of the user message module from the library assigned with SYSDIR-TASKLIB is not supported.
B B B B B B		When loading dynamically, the DBL searches for the user message module first in the library that you have assigned in LIB= . If this library does not exist, the DBL aborts the search. If the library exists but the user message module could not be found there, the DBL searches through the alternative libraries. The alternative libraries are those that have been assigned a file link name BLSLIB <i>nn</i> (0≤ <i>nn</i> ≤99).
B B B B B B	LOCALE=(<i>lang_id</i> , <i>terr_id</i>)	Language environment of the user-specific message modules defined by means of a language identifier and possibly a territorial identifier. By making the appropriate entries in the LOCALE= parameter of the USER or LTERM statement, you can assign a corresponding UTM message module. Messages are then output in the user's language.
B B B		If you issue more than one MESSAGE statement, each statement must contain the LOCALE= parameter. The <i>lang_id</i> and <i>terr_id</i> combination must be unique in each MESSAGE statement for a UTM message module.
B B	<i>lang_id</i>	Freely selectable language identifier for a UTM message module, up to two characters in length.
B		There is no default value for <i>lang_id</i> , i.e. this is a mandatory parameter.

B B	terr_id	Territorial identifier for a UTM message module up to two characters in length. You can also specify blanks for <i>terr_id</i> .
B B B B B B		If you specify MESSAGE ...,LOCALE=, you must also define the MAX ...,LOCALE (see page 385). The application message module of the UTM application is automatically the message module whose <i>lang_id</i> and <i>terr_id</i> in the MESSAGE statement match the locale in the MAX statement. openUTM uses the application message module for messages to SYSLST, SYSOUT and CONSOLE. The message destinations specified in the other message modules have no significance.
B B B B		The UTM message module whose <i>lang_id</i> and <i>terr_id</i> in the MESSAGE statement are identical to the values entered for LOCALE= in the USER or LTERM statement is used for messages to STATION, SYSLINE and PARTNER.
B B B B B		Specifications relating to the user have priority over those relating to the LTERM partner, i.e. if a user is signed on when a message is output, openUTM uses the UTM message module appropriate for that user. If the UTM message modules are assigned using language and territorial identifiers, the procedure is as follows:
B B B		– If a UTM message module exists with a <i>lang_id</i> and <i>terr_id</i> combination identical to the entries in the USER or LTERM statement, UTM messages are output in this language environment.
B B B		– If an identical combination cannot be found, the UTM message module with the same <i>lang_id</i> but for which no <i>terr_id</i> has been generated is used.
B		– If this is not possible, the application message module is used.

X/W **Generating message modules under Unix systems and Windows systems**

X/W Under Unix systems and Windows systems, you can generate exactly one user-defined message module with the MESSAGE statement, i.e. you may only specify the MESSAGE statement once within a single KDCDEF run.

X/W If a MESSAGE statement is not issued, the name of the external C/C++ structure is KCSMSGS. An object module with a C/C++ structure with this name is supplied with openUTM as a file.

X Under Unix systems, the file is the object module `kcsmsgs.o` in the library `utmpath/sys/libwork.a` or `utmpath/sys/libwork.so`.

W Under Windows systems the file `kcsmsgs.obj` in the directory `utmpath\sys`.

X/W MESSAGE_ MODULE=name

X/W **MODULE=name**

X/W Name of the external C/C++ structure with which messages are addressed. In the case of a user-specific message module (see the description of the KDCMMOD tool in the openUTM manual “Messages, Debugging and Diagnostics under Unix systems and Windows systems”), the name specified here must match the name of this module. *name* can be up to eight characters in length.

X/W This is a mandatory operand.

MPOOL - define a common memory pool (BS2000/OSD)


- B The MPOOL control statement allows you to define the properties of a common memory pool.
- B
- B The MPOOL statement can be issued several times, the only limit being the number of pools that can be created by a single process. Support is provided for up to eight common memory pools with SCOPE=GROUP or SCOPE=GLOBAL under a single user ID.
- B
- B The common memory pools are always created as FIXED. Every task that connects to an existing common memory pool is assigned the same address as the task that set up the common memory pool.
- B
- B The sequence of MPOOL statements within the generation run determines the order in which the common memory pools are created. Firstly, all common memory pools generated with SCOPE=GLOBAL are created in accordance with the sequence of MPOOL statements. This is followed by the creation of all common memory pools generated with SCOPE=GROUP, as defined by the sequence of MPOOL statements.

```

B MPOOL_      poolname
B             [ ,ACCESS={ READ | WRITE } ]
B             [ ,PAGE=X'xxxxxxxx' ]
B             [ ,SCOPE={ GROUP | GLOBAL } ]
B             ,SIZE=poolsize

```

- B poolname Name of the common memory pool. *poolname* must be unique within the UTM application and can be up to 50 characters in length.
- B
- B A number is appended to the name.
- B ACCESS= Access authorization
- B READ Read-only access to the common memory pool
- B Default: READ
- B WRITE Read and write access to the common memory pool

B B	PAGE=X'xxxxxxx'	Hexadecimal address in the format X'xxxxxxx'.
B B B		– 24-bit addressing mode: If the address is not a multiple of 64K (the four low-order half-bytes are 0), it is rounded off to a multiple of 64K.
B B B		– 31-bit addressing mode: The address is a multiple of 1MB. If this is not the case, it is rounded off to a multiple of 1MB.
B		Default:
B B		– 24-bit addressing mode: The pool is created starting with the lowest possible address.
B B B		– 31-bit addressing mode: The pool is created starting with the lowest possible address above X'01000000'.
B B B B B		If, in BS2000, global common memory pools are used in several UTM applications with the same contents/names, the parameter PAGE=X'xxxxxxx' must be specified with the same address in all applications. The address specified using PAGE= must be selected in such a way that the address area reserved is available in all these applications.
B B B		The common memory pools are always created as FIXED, i.e. all tasks of the UTM application find the pool at the same address in their virtual address space.
B B B B		An alternative to the use of PAGE= is to ensure that all the shared pools are generated in the same sequence in all applications. The MPOOL statements for shared pools must be specified at the beginning of the MPOOL statements.
B	SCOPE=	Scope of the memory pool
B	GLOBAL	All processes in the system
B	GROUP	All processes that run under the same user ID.
B		Default: GROUP
B B	SIZE=poolsize	Number of 64 KB memory segments in the pool (1 unit ≐ 64KB)
B B B		In 31-bit addressing mode, the memory segments are 1MB in length. The size of the common memory pool is thus rounded up to the nearest MB, which is calculated by multiplying <i>poolsize</i> by 64KB.
B		This is a mandatory operand.

MSG-DEST - define user-specific messages destinations

This statement allows you to define up to four additional user-specific message destinations for the UTM messages.

For this purpose, openUTM provides the unoccupied UTM message destinations, USER-DEST-1, USER-DEST-2, USER-DEST-3 and USER-DEST-4. MSG-DEST allows you to assign these UTM message destinations to concrete destinations. These destinations may be:

- a USER queue, or in other words, 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.

You can also assign several message destinations of the same type, for example, three LTERM partners and one USER queue. By defining the USER or TAC queue as the user-specific message destination you can ensure that the UTM messages are output to the WinAdmin administration workstation. More information can be found in the openUTM manual “Messages, Debugging and Diagnostics”.

```
MSG-DEST_    msgdest
              ,NAME=name
              ,DEST-TYPE={ LTERM | USER-QUEUE | TAC }
              [ ,MSG-FORMAT={ FILE | PRINT } ]
```

msgdest Name of the UTM message destination to which you wish to assign a user-specific message destination. Possible values are:

USER-DEST-1, USER-DEST-2, USER-DEST-3 or USER-DEST-4.

msgdest must also be assigned, using KDCMMOD, to the messages you wish to output to this user-specific message destination. For more information see [section “User-specific message destinations” on page 191](#) and the description of KDCMMOD in the openUTM manual “Messages, Debugging and Diagnostics”.

- NAME=name Name of the user-specific message destination. Possible values are:
- Name of a UTM user ID. This must be generated in a USER statement.
 - Name of an asynchronous TAC. This must be generated in a TAC statement with TYPE=A.
 - Name of a TAC queue. This must be generated in a TAC statement with TYPE=Q.
 - *BS2000/OSD*:
Name of an LTERM partner. This must be generated in an LTERM application and may not be assigned to a PTERM with PTYPE=UPIC-R.
 - *Unix systems and Windows systems*:
Name of an LTERM partner. This must be generated in an LTERM application and may not be assigned to a PTERM with PTYPE=UPIC-R or UPIC-L.

B
B
B
X/W
X/W
X/W
X/W

All messages that are linked via KDCMMOD to *msgdest* are then also output to the destination specified in *name*.



User-specific message destinations should not be locked or dynamically deleted because then no more messages will be output at this destination.

DEST-TYPE= Specifies the type of the message destination in *name*:

LTERM The message destination specified in *name* is an LTERM partner.

TAC The message destination specified in *name* is an asynchronous TAC or a TAC queue.

USER-QUEUE

The message destination specified in *name* is a USER queue.

MSG-FORMAT=

Specifies the format in which the message is passed to the message destination.

FILE The format corresponds to the data structures for the MSGTAC program. So only message inserts without messages texts are passed, the message inserts are not converted to a printable format.

PRINT The format corresponds to the output format of the UTM tool KDCPSYSL. So the message is prefixed with the date and time, followed by the message text with the text inserts and additional inserts. All inserts are printable.

KDCPSYSL is described in the openUTM manual “Using openUTM Applications”.

Default: FILE

MUX - define a multiplex connection (BS2000/OSD)

B The MUX control statement allows you to define the name and properties of a multiplex
 B connection between the UTM application and a Session Manager (OMNIS). This multiplex
 B connection can then be used simultaneously by several terminals to sign on to the UTM
 B application.

B The initiative for establishing the transport connection between openUTM and the Session
 B Manager can come from either side, but only the Session Manager can open a session.

B MUX_ name
 B [,BCAMAPPL=local_appliname]
 B [,CONNECT={ Y | N }]
 B [,MAXSES=number]
 B [,NETPRIO={ MEDIUM | LOW }]
 B ,PRONAM={ processorname | C'processorname' }
 B [,STATUS={ ON | OFF }]

B name Name of the multiplex connection



The specified name must be unique and must not be assigned to any other object in name class 3. See also [section "Uniqueness of names and addresses" on page 265](#).

B BCAMAPPL=local_appliname

B Local name of the UTM application as defined in the MAX statement
 B (APPLINAME on [page 372](#)) or BCAMAPPL statement (see [page 291](#)). This
 B name is then used to establish a connection to the Session Manager, i.e.
 B the Session Manager must specify *local_appliname* as the partner name
 B when connecting to the UTM application. By issuing several MUX state-
 B ments with different BCAMAPPL names, you can set up parallel connec-
 B tions to the Session Manager.

B The BCAMAPPL name specified in the CLUSTER statement is not
 B permitted here.

B Default:

B Application name defined in the statement MAX APPLINAME=*appliname*

B	CONNECT=	Set up the local transport connection on application start
B	Y	When starting the application, openUTM attempts to establish a logical transport connection to the Session Manager.
B		If unsuccessful, openUTM repeats its attempt to establish the connection at intervals defined in MAX ...,CONRTIME= <i>time</i> .
B		Default: Y
B	N	When starting the application, openUTM does not attempt to establish a connection to the Session Manager.
B	MAXSES=number	
B		Maximum number of simultaneously active sessions between the Session Manager and the UTM application
B		
B		
B		
B		openUTM creates <i>number</i> LTERM partners internally for the specified number of sessions. The number of LTERM partners must be taken into consideration in the maximum number of UTM names. See section “Maximum values for names” on page 263 .
B		Default value: 10
B		Minimum value: 1
B		Maximum value: 65000 (theoretical value)
B	NETPRIO=	Transport priority to be used on the transport connection between the Session Manager and the UTM application
B		Default: MEDIUM
B	PRONAM={ processorname C'processorname' }	
B		Name of the system on which the Session Manager is located.
B		If the <i>processorname</i> contains special characters it must be entered as a character string using C'...'
B	STATUS=	Status of the multiplex connection
B	ON	The connection to the Session Manager is not locked.
B		Default: ON
B	OFF	The connection to the Session Manager is locked. A connection cannot be established between the Session Manager and the UTM application.
B		This status can be modified by the administrator.

OPTION - manage the KDCDEF run

The OPTION control statement allows you to manage the KDCDEF run. openUTM can only read the KDCDEF options from a procedure file or shell script; the actual generation statements are read from other files (under BS2000/OSD from SAM or ISAM files).

OPTION statements are read from SYSDTA or *stdin* only.

OPTION statements are ignored by KDCDEF, if it is read from a file that is assigned using OPTION DATA=.

If you issue more than one OPTION statement, the values last specified are taken as valid.

If the OPTION statement is not specified, only the KDCFILE is generated, i.e. the default setting GEN=KDCFILE applies.

```
OPTIONL      [ ,DATA=filename ]
              [ ,GEN= { KDCFILE | ROOTSRC | NO | ALL |
                        CLUSTER
                        (KDCFILE,ROOTSRC)
                        (CLUSTER,KDCFILE)
                        (CLUSTER,ROOTSRC)
                        (CLUSTER,KDCFILE,ROOTSRC) } ]
              [ ,GEN-RSA-KEYS={ YES | NO } ]
```

B *BS2000/OSD-specific operand*

B [,ROOTSRC=filename]

X/W *further operand for Unix systems and Windows systems*

X/W [,CHECK-RFC1006={ NO | YES }]

X/W CHECK-RFC1006=

X/W Extended check of the UTM generation for the communication via TCP/IP
X/W connections with RFC1006.

X/W YES KDCDEF checks the specifications of transport addresses for all communi-
X/W cation partners and local transport system end points that are generated
X/W with T-PROT= RFC1006 for completeness and plausibility. When OPTION
X/W CHECK-RFC1006=YES, a port number must be specified in the
X/W LISTENER-PORT parameters of the ACCESS-POINT, BCAMAPPL, CON,
X/W OSI-CON, and PTERM statements.

X/W Default: YES

X/W NO KDCDEF does not execute any extended checks.

DATA=filename

The KDCDEF control statements are read from the file specified here (under BS2000/OSD a SAM or ISAM file). When the end of the file is reached, the next KDCDEF control statements are read from SYSDTA or *stdin*.

You use the `DATA=control_statements_file` statement to assign KDCDEF input files which are created with the CREATE-CONTROL-STATEMENTS statement. The `control_statements_file` input files contain configuration information which is read from the current KDCFILE and converted into KDCDEF control statements.

For information on the inverse KDCDEF function, see [section “Inverse KDCDEF” on page 268](#).

GEN= Specifies what objects are to be generated.

KDCFILE The KDCFILE is generated.

In the case of a UTM cluster application, the cluster user file must already exist. This is evaluated and extended if required.

Default: KDCFILE

ROOTSRC

The ROOT table source is generated.

(KDCFILE,ROOTSRC)

The KDCFILE and the ROOT table source are generated.

In the case of a UTM cluster application, the cluster user file must already exist. This is evaluated and extended if required.

CLUSTER

The following UTM cluster files are generated:

- the cluster configuration file
- the cluster user file
- the cluster page pool files
- the cluster GSSB file
- the cluster ULS file

These files must not already exist.



If you have specified OPTION GEN=CLUSTER or (CLUSTER,...), you must also specify a CLUSTER statement and at least two CLUSTER-NODE statements.

(CLUSTER,KDCFILE)

The UTM cluster files listed above are generated together with the KDCFILE.

(CLUSTER,ROOTSRC)

The UTM cluster files listed above are generated together with the ROOT table source.

(CLUSTER,KDCFILE,ROOTSRC)

The UTM cluster files listed above are generated together with the KDCFILE and the ROOT table source.

NO The parameters are only checked.

ALL The KDCFILE and the ROOT table source are generated.



If a ROOT table source is generated, the ROOT statement must be specified. This is the case with the following specifications:

- ROOTSRC
- ALL
- (CLUSTER,ROOTSRC)
- (CLUSTER,KDCFILE,ROOTSRC)

X/W

The following issues must be noted for Unix systems / Windows systems:

X/W

- KDCDEF generates the ROOT table source as a C/C++ program under the name *rootname.c* (see the ROOT statement) in the directory *filebase* (see the MAX ...,KDCFILE=*filebase*... statement).

X/W

X/W

GEN-RSA-KEYS =

Specifies whether RSA keys are to be created.

This parameter is relevant only if objects with encryption were generated in the application, see ENCRYPTION-LEVEL operand for the TAC, PTERM and TPOOL statements.

YES RSA keys are created if objects that require encryption were generated in the application.

Default.

NO No RSA keys are created.

GEN-RSA-KEYS=NO should not be used unless, after the KDCDEF run, the required RSA keys are transferred from an old KDCFILE to the new KDCFILE using KDCUPD. For more information on transferring RSA keys with KDCUPD see [page 577](#).

If objects with encryption are generated in an application and if no RSA keys are available, the application can run but with certain restrictions, i.e.:

- TACs with encryption cannot be called,
- no connection can be established to PTERMs or TPOOLS generated with encryption.

- B** ROOTSRC=filename
- B** This parameter is significant only when generating the ROOT table source.
- B** *filename* can be up to 54 characters in length.
- B** A ROOT source with the CSECT name *rootname* is generated and stored in the KDCROOT file *filename*. *rootname* is defined in the ROOT statement.
- B** Default: ROOT.SRC.ASSEMB.*rootname*

OSI-CON - define a logical connection to an OSI TP partner

The OSI-CON control statement allows you to assign a real partner application to an OSI-LPAP partner for communication based on the OSI TP protocol. It is used to define logical connections between the local UTM application and a partner application. For this purpose, you must specify:

- the name of the OSI TP access point in the local application, via which the connection is to be established. This is defined using the ACCESS-POINT statement.
- the address of the OSI TP access point of the partner application. This address consists of a P-selector, an S-selector, a T-selector and an N-selector.

X/W
X/W

Under Unix systems and Windows systems the following operands are used to describe the T-selector:

X/W
X/W
X/W
X/W
X/W

- TRANSPORT-SELECTOR (=address of the partner application on the partner computer)
- T-PROT (the transport protocol used)
- TSEL-FORMAT (format identifier of the T-selector)
- LISTENER-PORT (port number for RFC1006)

X/W
X/W

See [section “Providing address information for the CMX transport system \(Unix systems and Windows systems\)” on page 110ff](#) for more information.

The partner application sets up the connection to the local application via an OSI-LPAP partner, which is defined in the OSI-LPAP statement. Here you generate the number of connections, the names of the individual connections, and so on. The communication parameters of the OSI-LPAP partner are assigned to the OSI-CON statement using the operand `OSI-LPAP=osi_lpap_name`. The logical connection is thus generated in a single OSI-CON statement, even if there are several parallel connections to the partner application.

If a partner application can be accessed in various remote systems at different times, you must define several addresses and thus replacement connections for the OSI-LPAP partner assigned to this application. For generation purposes, this involves assigning several OSI-CON statements (OSI-CON statements with the same *osi_lpap_name* and LOCAL-ACCESS-POINT) to a single OSI-LPAP statement (see [page 426](#)). However, only one OSI-CON statement can be active at any one time. You can switch to a replacement connection by means of administration.

When you use OSI-LPAP bundles, then the following also applies to the OSI-CONs of the slave LPAPs in a LPAP bundle:

All OSI-CONs of all slave LPAPs in a LPAP bundle must be assigned the same access point (see also [section “MASTER-OSI-LPAP - Defining the master LPAP of an OSI-LPAP bundle” on page 367](#)“).

```

OSI-CON_  connection_name
          [ ,ACTIVE={ YES | NO } ]
          ,LOCAL-ACCESS-POINT=access_point_name
          ,NETWORK-SELECTOR=C'c'
          ,OSI-LPAP=osi_lpap_name
          ,PRESENTATION-SELECTOR={ *NONE |
                                   (C'c' [ , STD | EBCDIC | ASCII ]) |
                                   X'xx' }
          ,SESSION-SELECTOR={ *NONE |
                               (C'c' [ , STD | EBCDIC | ASCII ]) |
                               X'xx' }
          ,TRANSPORT-SELECTOR=C'c'

```

X/W

further operands for Unix systems and Windows systems

X/W

[,LISTENER-PORT=number]

X/W

[,MAP={ USER | SYSTEM }]

X/W

[,T-PROT={ RFC1006]

X/W

[,TSEL-FORMAT={ T | E | A }]

connection_name

Name of the logical connection between the local UTM application and the partner application for communication based on the OSI TP protocol.

connection_name identifies the connection in the local application. It can be up to eight characters in length and must be unique in the local application.

ACTIVE=

Status (active or inactive) of the logical connection to the partner application. In the case of replacement connections to the partner application, several OSI-CON control statements are issued with the same *osi_lpap_name* of an OSI-LPAP partner. However, only one OSI-CON statement can be generated with ACTIVE=YES. All others must be defined with ACTIVE=NO. You can then switch to the replacement connections by means of administration.

YES

The connection defined in this OSI-CON statement is active.

Default: YES

NO

The connection defined in this OSI-CON statement is inactive.

X/W	LISTENER-PORT=number	
X/W		Port number of the partner application if the connection to the partner application is to be established via TCP/IP.
X/W		Permitted values: 102 and 1025 - 32767
X/W		Default: 0 (no port number)
X/W		When OPTION CHECK-RFC1006=YES, a port number must be specified for the LISTENER-PORT.
X/W		
	LOCAL-ACCESS-POINT=access-point_name	
		Name of the local OSI TP access point used for communication with the partner application. This is defined using the ACCESS-POINT control statement.
		If replacement connections (several OSI-CON statements with the same <i>osi_lpap_name</i>) have been defined for the OSI-LPAP partner to which the partner application is assigned, the same local access point must be specified for all replacement connections.
		If the application context of the OSI-LPAP partner uses the CCR syntax, the following address components must also be defined for the local access point:
		– APPLICATION-ENTITY-QUALIFIER (see the description of the ACCESS-POINT statement on page 277)
		– APPLICATION-PROCESS-TITLE (see the description of the UTMD statement on page 544)
X/W	MAP=	This controls ASCII/EBCDIC conversion when exchanging unformatted messages with partner applications.
X/W		Formatted messages (where KCMF contains a format identifier) are generally not subject to message handling by openUTM.
X/W	USER	openUTM does not perform message handling, i.e. the data in the KDCS message area is transferred to the partner application unchanged.
X/W		Default: USER
X/W	SYSTEM	openUTM converts the data of the message area from ASCII to EBCDIC before sending and from EBCDIC to ASCII after receiving. The message may only contain printable characters and must be created in line mode (KCMF = blank).
X/W		
X/W		
X/W		

NETWORK-SELECTOR=C'c'

Name of the partner computer. The name may be up to 8 characters long.

N-SEL= is a mandatory operand.

B

BS2000/OSD:

B

In N-SEL= you must specify the BCAM processor name of the system on which the partner application is located.

B

X/W

Unix systems and Windows systems:

X/W

There are two options for specifying N-SEL:

X/W

– You enter the real host name under which the IP address of the partner computer is entered in the name service of the local system (e.g. the hosts file). You must not specify alias names of the computer.

X/W

X/W

X/W

– You enter the UTM host name of the partner computer. This is only possible when you have set the UTM_NET_HOSTNAME environment variable and specified the UTM host name in the conversion file (see the [section “Using mapped host names \(Unix systems and Window systems\)”](#) on page 122).

X/W

X/W

X/W

X/W



Please note that the name pair (TRANSPORT-SELECTOR, NETWORK-SELECTOR) specified here must not be identical to the name pair (*remote_applname*, PRONAM) defined in a CON statement ([page 313](#)), or to the name pair (*ptermname*, PRONAM) defined in a PTERM statement ([page 437](#)).

OSI-LPAP=osi-lpap_name

Name of the OSI-LPAP partner defined as the logical access point for the partner application in the local application.

osi_lpap_name must be defined in a OSI-LPAP statement.

osi_lpap_name can be up to eight characters in length.

PRESENTATION-SELECTOR=

Presentation selector of the partner application. This is the address component of the OSI TP access point in the remote partner's system. The specified value must match the presentation selector defined for this access point in the partner application.

*NONE A symbolic presentation selector is not defined.

C'c' The presentation selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The presentation selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted:

STD The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix systems and Windows systems = ASCII).

Default: STD

EBCDIC

The characters are interpreted as EBCDIC code.

ASCII The characters are interpreted as ASCII code.

X'x' The presentation selector is entered in the form of a hexadecimal number (x). The value specified for *x* can be up to 32 hexadecimal digits (\cong 16 bytes) in length. You must enter an even number of hexadecimal digits.

SESSION-SELECTOR=

Session selector of the partner application. This is the address component of the OSI TP access point in the remote partner's system. The specified value must match the session selector defined for this access point in the partner application.

*NONE A session selector is not defined.

C'c' The session selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The session selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted:

STD The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix systems and Windows systems = ASCII).

Default: STD

EBCDIC

The characters are interpreted as EBCDIC code.

ASCII The characters are interpreted as ASCII code.

X'x' The session selector is entered in the form of a hexadecimal number (x). The value specified for *x* can be up to 32 hexadecimal digits (\cong 16 bytes) in length. You must enter an even number of hexadecimal digits.

TRANSPORT-SELECTOR=C'c'

Transport selector of the partner application.

You can enter up to eight printable characters. Permitted characters include uppercase letters, numbers, and the special characters \$, # and @. Hyphens are not permitted. The first character must be an uppercase letter.

T-SEL= is a mandatory operand.

In T-SEL= you must specify the following:

B
B
X/W
X/W
X/W
X/W

- *BS2000/OSD*:
The BCAM application name of the remote partner.
 - *Unix systems and Windows systems*:
T-selector of the partner application.
- You must specify the T-selector in T-SEL that is assigned to the partner application in the remote system for CHECK-RFC1006=YES.



Please note that the name pair (TRANSPORT-SELECTOR, NETWORK-SELECTOR) specified here must not be identical to the name pair (*remote_appliname*, PRONAM) defined in a CON statement (on [page 313](#)), or to the name pair (*ptermname*, PRONAM) defined in a PTERM statement (on [page 437](#)).

X/W
X/W
X/W
X/W
X/W
X/W
X/W

T-PROT= The address format with which the OSI TP partner signs on to the transport system.

Information on the following address formats can be found in the "CMX User Guide" and in the PCMX online help system.

RFC1006 Address format RFC1006
ISO transport protocol based on TCP/IP and RFC1006 convergence protocol.

Default: RFC1006

X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W
X/W

TSEL-FORMAT=
The format identifier of the T-selector
The format identifier specifies the coding of the T-selector in the transport protocol. You will find more information in the "CMX User Guide" and in the PCMX online help system.

T TRANSDATA format

E EBCDIC format

A ASCII format

Default:

T If the character set of the value of T-SEL corresponds to the TRANSDATA format

E Otherwise

It is recommended to explicitly specify a value for TSEL-FORMAT operation via TCP/IP with RFC1006.

OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP

The OSI-LPAP control statement allows you to define a logical access point in the local application for a partner application with which you wish to communicate on the basis of the OSI TP protocol. This logical access point is known as an OSI-LPAP partner. For each OSI-LPAP partner, you must define a logical partner name and the following logical connection properties:

- the application entity title (AET) of the partner application. This must be defined if you are working with transaction management (commit functional unit) or if a heterogeneous partner requires an AET in order to establish a connection. The AET consists of the following components, which must be specified for the partner application:
 - the application entity qualifier (AEQ) of the remote access point (see the description of the ACCESS-POINT statement on [page 277](#))
 - the application process title (APT) of the partner application (see the description of the UTMD statement on [page 544](#))
- the application context used for communication with the partner application based on the OSI TP protocol. If you are not using a standard application context, you define your application context using the APPLICATION-CONTEXT statement (see [page 285](#)). If the application context of the OSI-LPAP partner contains the CCR syntax, an AEQ and an APT must be specified for the partner application.
- the number and properties of connections to the partner application
- access rights of the partner application in the local application
The operands KSET and ASS-KSET are provided to define the access rights. In KSET you specify the highest level of access rights of the OSI TP partner that the OSI TP partner will have when it signs on to the local application with a user ID. You can restrict these access rights with the ASS-KSET operand. The restricted access rights take effect when the OSI TP partner does not pass a user ID when signing on, i.e. the "association user" is active.
- administration authorization of the partner application in the local application
- maximum values for the message queue of the OSI-LPAP partner.

If a communication partner can be accessed in various remote systems at different times, you can assign several addresses to this partner. This involves assigning several OSI-CON statements (with the same *osi_lpap_name*, see [page 420](#)) to a single OSI-LPAP statement. However, only one OSI-CON statement can be active at any one time. You can switch to a replacement connection by means of administration. All OSI-CON connections belonging to an OSI-LPAP partner must have the same local access point.

You can generate a maximum of 21000 associations.

```

OSI-LPAP_  osi_lpap_name
            ,APPLICATION-CONTEXT=application_context_name
            [ ,APPLICATION-ENTITY-QUALIFIER=application_entity_qualifier
            ,APPLICATION-PROCESS-TITLE=object_identifier ]
            [ ,ASS-KSET=keysetname2 ]
            ,ASSOCIATION-NAMES=association_name
            [ ,ASSOCIATIONS=number ]
            [ ,BUNDLE=master-lpap-name]
            [ ,CONNECT=number ]
            ,CONTWIN=number
            [ ,IDLETIME=time ]
            [ ,KSET=keysetname1 ]
            [ ,PERMIT={ ADMIN | SATADM1 | ( ADMIN,SATADM )1 } ]
            [ ,QLEV=number ]
            [ ,STATUS={ ON | OFF } ]
            [ ,TERMN=termn_id ]

```

B ¹ only permitted under BS2000/OSD

osi_lpap_name

Name of the OSI-LPAP partner of the partner application, which is used by the program units of the local UTM application to address the partner application. *osi_lpap_name* can be up to eight characters in length.

osi_lpap_name must be unique and must not be assigned to any other object in name class 1. See also [section “Uniqueness of names and addresses” on page 265](#).

APPLICATION-CONTEXT=application_context_name

Name of the application context to be used by the partner application.

This is a mandatory operand.

By default, openUTM supports the following application contexts:

- UDTAC
- UDTDISAC
- XATMIAC
- UDTCCR
- UDTSEC
- XATMICCR

Further information can be found in the description of the APPLICATION-CONTEXT statement on [page 285](#). If the generated application context does not match that used by the partner application, openUTM rejects the connection request with the following UTM messages:

```
P001 APPLICATION CONTEXT NOT SUPPORTED    or
P011 Abstract syntax not permitted
```

APPLICATION-ENTITY-QUALIFIER=application_entity_qualifier

Application entity qualifier of the partner application. This is combined with the application process title to address a partner application when using a heterogeneous link or working with transaction management (commit functional unit). *application_entity_qualifier* is a positive integer used to call the partner application in the remote system.

If the application context contains the CCR syntax, this is a mandatory operand. The name pair *application_entity_qualifier* and *object_identifier* must be unique within the UTM application.

The *application_entity_qualifier* specified here must be assigned to access point in the partner application.

Minimum value: 1

Maximum value: 67108 863 ($2^{26}-1$)

APPLICATION-PROCESS-TITLE=object_identifier

The application process title of the partner application is to be specified as the *object_identifier*. The application process title is combined with the application entity qualifier to address a partner application when using a heterogeneous link or working with transaction management (commit functional unit).

If the application context contains the CCR syntax, this is a mandatory operand. The name pair *application_entity_qualifier* (of the OSI-LPAP statement) and *object_identifier* must be unique within the UTM application.

For information on defining the APT, see the UTMD statement on [page 544](#).

ASS-KSET=ksetname2

ASS-KSET is only allowed if the local application is generated with user IDs. You may only set ASS-KSET in conjunction with KSET.

You must specify the name of the key set in *ksetname2*. The key set must be defined with a KSET statement.

You specify the minimum access rights that the partner application can have in the local application with ASS-KSET= .

The key set specified in *ksetname2* takes effect when the partner application does not pass a user ID to openUTM when establishing the association. The access rights result from the set of key codes contained in the key set generated with KSET= and with ASS-KSET= (intersection of the sets). For this reason, all key codes contained in ASS-KSET=*ksetname2* should also be contained in KSET=*ksetname1*.

Default: No key set

The access rights specified in KSET are always valid.

ASSOCIATION-NAMES=association_name

Name defined in the local application for logical connections to the partner application.

Connection names consists of the value of *association_name* as a prefix, followed by a serial number between 1 and the value of the ASSOCIATIONS operand, i.e. the number of parallel connections. The entire name can be up to eight characters in length. The maximum length of *association_name* depends on the value specified for ASSOCIATIONS. The following applies for the number of connections:

Number of decimal places in the value specified for ASSOCIATIONS + number of characters in *association_name* ≤ 8

Example

If ASSOCIATIONS=10 and
ASSOCIATION-NAMES=ASSOC,

The connection names are ASSOC01, ASSOC02,...,ASSOC10.

These are used as association names in the local UTM application. The same name must not be defined for a user ID (USER) or a session name for distributed processing based on LU6.1 (LSES).

ASSOCIATIONS=number

Maximum number of parallel connections to the partner application. This depends on layers 1 - 6 of the OSI reference model defined by ISO (in particular on layer 4, the transport layer).

The number of parallel connections must be coordinated with the generation of the partner application.

Default: 1

Minimum value: 1

Maximum value: The maximum number of associations is restricted by the size of the name space of the UTM application (see [section "Number of names" on page 262](#)).

BUNDLE=master-lpap-name

Name of a master LPAP. By specifying *master-lpap-name*, this OSI-LPAP partner becomes a slave LPAP of the corresponding master LPAP.

The master LPAP specified here must be generated with a MASTER-OSI-LPAP statement.

CONNECT=number

Number of connections to be established automatically with the partner application when the local application is started. Automatic connection setup can be requested in either the local application or the partner application. The connection is established as soon as both partners are available.

Default: 0

Maximum value: Number of parallel connections specified in the ASSOCIATIONS operand.

CONTWIN=number

Number of connections for which the local application is to act as the contention winner. The local application is the contention loser for all other connections (ASSOCIATIONS= entry minus CONTWIN= entry).

The contention winner of a connection is responsible for managing that connection. However, jobs can be started both by the contention winner and by the contention loser. If both communication partners attempt to initiate a job simultaneously, the connection is reserved by the contention winner job.

This is a mandatory operand.

The number of contention winners and contention losers must be coordinated with the generation of the partner application.

The contention winner should be the communication partner that initiates jobs most frequently.

Minimum value: 0

Maximum value: Number of parallel connections specified for the ASSOCIATIONS operand.

IDLETIME=time

Number of seconds for which the idle state of a connection is monitored. If the connection is not reserved by a job within the period specified in *time*, openUTM shuts down the connection.

IDLETIME=0 means that the idle state of the connection is not monitored.

Default: 0

Minimum value: 0

Maximum value: 32767

KSET=keysetname1

Specifies the maximum access rights of the partner application in the local application. The name of a key set is to be specified in *keysetname1*. The key set must be defined with a KSET statement.

If the OSI TP partner does not pass a user ID to the local application for an OSI TP dialog, then its access rights for this OSI TP dialog result from the set of key codes that are in the key set generated with KSET= as well as with ASS-KSET= (intersection).

The key set *keysetname1* should therefore also contain all key codes that are in the key set generated with ASS-KSET= .

If the OSI TP partner does pass a user ID, then its access rights for this OSI TP dialog result from the set of key codes that are contained in the key set of the user ID as well as in the key set of the OSI-LPAP generated with KSET.

Default: No key set,
i.e. only those services can be started or remote services (LTAC) generated in the local application can be addressed that are not secured with a lock code.

PERMIT= Authorization level of the partner application

ADMIN The partner application can execute administration functions in the local application.

B
B
B **SATADM** The partner application can execute SAT preselection functions in the local application, i.e. it can activate and deactivate the SAT logging of certain events (UTM SAT administration authorization).

B
B
B **(ADMIN,SATADM)** The partner application can execute administration and SAT preselection functions in the local application.

Default:

If the operand is not specified, the partner application cannot execute administration and SAT preselection functions in the local application.

QLEV=queue_level_number

Maximum number of asynchronous messages that can be accommodated in the message queue of the OSI-LPAP partner. If this threshold value is exceeded, further APRO-AM calls to this LPAP partner are rejected with UTM message 40Z.

Default: 32767

Minimum value: 0

Maximum value: 32767 (i.e no restriction of the queue length)

- STATUS=** Specifies whether the OSI-LPAP partner is locked. This status can be modified by the administrator using the KDCLPAP administration command.
- ON** The OSI-LPAP partner is unlocked. Connections between the partner application and the local application can be established or may be already in place.
Default: ON
- OFF** The OSI-LPAP partner is locked. Connections cannot be established between the partner application and the local application.
- TERMN=termn_id** Identifier up to two characters in length, which indicates the type of communication partner. *termn_id* is not queried by openUTM, but is used by the user when querying or grouping terminal types, for example. *termn_id* is entered in the KB header for services, i.e. for services started by the partner application in the local application.
Default: A6

PROGRAM - define a program unit

The PROGRAM control statement allows you to define the name and properties of a program unit.

If a ROOT table source is to be generated in the KDCDEF run (OPTION statement with GEN=ROOTSRC or GEN=ALL), then you must issue at least one PROGRAM statement.

B Generating UTM program units under BS2000/OSD

```

B PROGRAM_      objectname
B
B              , COMP={ ASSEMB |
B                  C |
B                  COB1 |
B                  FOR1 |
B                  PASCAL-XT |
B                  PLI1 |
B                  SPL4 |
B                  ILCS }
B
B              [ ,LOAD-MODULE=1modname ]

```

B objectname Access point for a program unit (CSECT or ENTRY name). *objectname* may
B be up to 32 characters in length.

B For details on the characters allowed refer to the [section "Format of names"](#)
B [on page 260](#).


B COMP= Designates the runtime system that the program unit will be used for.

B This is a mandatory operand.

B You must specify COMP=ILCS for all program units that support ILCS (Inter
B Language Communication Services), e.g. program units under COBOL85,
B FORTRAN90, C, etc.

B Whether or not ILCS is supported depends on the compiler version used
B and on the runtime system version under which the program unit runs.

B The value that you must specify for COMP can be found in the appendix of
B the openUTM manual "Using openUTM Applications under BS2000/OSD".
B Please consider these notes especially if the programs were compiled with
B an older compiler version.

B  COMP=C is a synonym for COMP=ILCS

B The KDCADM administration program must be generated with
B COMP=ILCS and the KDCSHUT transaction code must be assigned at
B least with a TAC statement.

B	LOAD-MODULE= <i>lmodname</i>
B	LOAD-MODULE identifies the name of the load module in which the
B	program unit was linked. This load module must be defined using the LOAD-
B	MODULE statement. <i>lmodname</i> can be up to 32 characters in length.
B	This name is subject to the same rules as the element names of a program
B	library (see also section “Format of names” on page 260).
B	Please note the following when using the LOAD-MODULE operand:
B	– The KDCADM administration program must not be assigned to a load
B	module generated with LOAD-MODE=ONCALL in the LOAD-MODULE
B	statement.

X/W Generating UTM program units under Unix systems or Windows systems

X/W	PROGRAM_	objectname
X/W		,COMP={ <u>C</u> COB2 CPP }
X/W		[,SHARED-OBJECT=shared_object_name]

X/W	objectname	Name of the access point of the program unit. The name must be alphanu-
X/W		meric and may be up to 32 characters in length. For details on the
X/W		characters allowed refer to the section “Format of names” on page 260 .
X/W	COMP=	Designates the compiler used to compile the program unit.
X/W	C	C compiler
X/W		Default: C
X/W	CPP	C++ compiler
X/W	COB2	COBOL compiler (Server Express / NetExpress)
X		Under Unix systems this generates a Microfocus COBOL program that was
X		created by Server Express from MicroFocus.
X		Only numbers and uppercase letters can be used for the PROGRAM-ID and
X		the access points. This not only complies with IBM conventions, but also
X		guarantees the portability of the programs.
W		Under Windows systems this generates a COBOL program that was
W		created by NetExpress from MicroFocus.

X/W SHARED-OBJECT=shared_object_name
X/W (Program exchange using the dynamic linker)
X/W This operand need only be specified if the program unit is to be loaded
X/W dynamically. *shared_object_name* is the name of the shared object (Unix
X/W system) or DLL (Windows system) into which the program unit was incorpo-
X/W rated. This shared object/DLL must be defined using the SHARED-
X/W OBJECT statement.

PTERM - define the properties of a client/printer and assign an LTERM partner

The PTERM control statement allows you to define the properties of a physical client or printer of the UTM application.

Clients are terminals, UPIC clients and transport system applications. Transport system applications (for short TS application) are understood to be all applications that are generated as PTYPE=APPLI or PTYPE=SOCKET. See also the PTYPE operand in the table on [page 451](#) (BS2000/OSD) or [page 456](#) (Unix systems and Windows systems).

You must always issue a PTERM statement for clients when connections to the client or printer are to be established from the local UTM application.

The PTERM statement allows you to assign an LTERM partner defined using the LTERM statement to the client/printer. A separate LTERM statement must be written for each client or printer (see also [page 355](#) for more information on this subject).

If desired, you can first define the client/printer in a PTERM statement and then assign it to an LTERM partner later on during operation by means of dynamic administration. Exceptions are UPIC clients and TS applications. You need to assign an LTERM partner to these immediately.

If LTERM pools have not been generated (TPOOL statement, see [page 511](#)), you must assign a client in the LTERM= operand of at least one PTERM statement. Only then can a connection be established in order to access the application.

W
W



Printers are not supported by openUTM under Windows systems.

Address of the client or printer

For the application to be able to establish connections to the partner application, you must specify the partner address. The following operands are used to do this:

- *ptermname* (name/T-selector of the communication partner)
- PRONAM (name of the host partner) ¹⁾
- LISTENER-PORT (TCP/IP port number).²⁾

X/W
X/W
X/W

1) Under Unix systems and Windows systems, the name of the partner processor may only be specified if the partner is a remote UPIC client (UPIC-R) or a TS application (PTYPE= APPLI or SOCKET).

B

2) Under BS2000/OSD, LISTENER-PORT may only be specified with PTYPE=SOCKET.

X/W The following operands are used for further definition of the partner address under Unix
X/W systems and Windows systems:

X/W – T-PROT (address format for the transport protocol used)

X/W – TSEL-FORMAT (format identifier of the T-selector)

X/W See [section “Providing address information for the CMX transport system \(Unix systems
X/W and Windows systems\)” on page 110ff](#) for more information
X/W or [section “Providing address information for the SOCKET transport system \(Unix systems
X/W and Windows systems\)” on page 118](#).

If the connection to a TS application is to be established via the socket interface with native TCP/IP as the transport protocol, then you must specify the computer on which the TS application will run in PRONAM and the port number on the host partner on which the TS application waits for requests to establish a connection from the network in LISTENER-PORT. You must specify an application name that was generated for T-PROT=SOCKET in BCAMAPPL (see also [page 291](#)).

You can specify whether or not openUTM is to handle messages in the MAP operand.

Uniqueness of names

When generating the CON, PTERM and MUX statements, please note that the name triplet (*appliname* or *ptermname*, *processorname*, *local_appliname*) must be unique within the generation run.

B
B
B
B
B
B



In order to make the generation of your UTM application more independent of the PDN generation, it is possible to incorporate terminals in the configuration without explicitly specifying their type. For this purpose, set the PTYPE operand to *ANY. During connection setup, openUTM then takes the partner type (PTYPE) from the user services protocol (connection letter) and checks whether or not this type is supported. If not, openUTM rejects the connection request.

```

PTERM_      ptermname
            [ ,BCMAPPL=local_applname ]
            [ ,CONNECT={ YES | NO } ]
            [ ,ENCRYPTION-LEVEL={ NONE | 1 | 2 | 3 | 4 | TRUSTED } ]
            [ ,IDLETIME=time ]
            [ ,LISTENER-PORT=number ]
            [ ,LTERM=ltermname ]
            [ ,MAP={ USER | SYSTEM | SYS11 | SYS21 | SYS31 | SYS41 } ]
            ,PRONAM= { processorname | C'processorname' | *RSO1
            }
                only mandatory in BS2000/OSD
            [ ,STATUS={ ON | OFF } ]
            [ ,TERMN=termn_id ]
            [ ,USP-HDR={ NO | MSG | ALL } ]

```

B/X

BS2000/OSD and Unix system specific operand

B/X

[,CID=printer_id]

B

BS2000/OSD specific operands

B

[,PROTOCOL={ N | STATION }]

B

,PTYPE={ partnertyp | *ANY | *RSO }

B

[,USAGE={ D | 0 }]

X/W

Unix system and Windows system specific operands

X/W

[,PTYPE={ partnertyp |
PRINTER² | (PRINTER ,printertype [,class])² }]

X/W

[,T-PROT=RFC1006 | SOCKET]

X/W

[,TSEL-FORMAT={ T | E | A }]

B

¹ These operand values are only permitted under BS2000/OSD.

X

² These operand values are only permitted under Unix systems

ptermname Name of the client or printer up to 8 characters in length

The specified name must be unique and must not be assigned to any other object in name class 3. See also [section “Uniqueness of names and addresses” on page 265](#)

The following cases can arise:

Socket applications (PTYPE=SOCKET)

- If the connection is to be established from the local application to the client, then any *ptermname* can be selected. It is only relevant internally in UTM then, e.g. for administration.
- If the connection is to be established externally (initiated by the client), then *ptermname* must contain the port number via which the client addresses the UTM application. You must then specify the prefix “PRT” followed by 5 digits (with leading zeros, if necessary) that designate the port number as the *ptermname*. For example, you must specify *ptermname*=PRT08050 if the client is to address the UTM application via the port 8050.

Establishing the connection externally to a specific PTERM is only possible by partners that set their port numbers themselves when establishing a connection. openUTM does not do this, i.e. you cannot issue any PTERM statements for a remote UTM application that is to establish SOCKET connections to the local application. In this case, you need to connect via an LTERM pool.

B
B

Clients (≠ socket applications) and printers connected to a application under BS2000/OSD

B
B

You must specify the name of client or printer defined during generation of the network. Please consult your network administrator.

B
B

When defining an RSO printer (PTYPE=*RSO), you must specify the name of the printer as defined for RSO.

X/W
X/W

UPIC clients and TS applications (≠ socket applications) connected to a UTM application under Unix systems or Windows systems

X/W
X/W

For OPTION CHECK-RFC1006=YES you must specify the T-selector that is assigned to the client in the remote system for *ptermname*.

X

Printers connected to a UTM application under Unix systems

X

In the case of printers, *ptermname* is the name of the spool queue or printer group as defined during generation of the Unix system.

X

To output the data, the printer process (*utmprint*) calls the *utmlp* script (see PTYPE=PRINTER on [page 453](#)).

X

X

Terminals connected to a UTM application under Unix systems

X

In the case of local terminals and pseudo terminals, the result of the command `basename `tty`` must be specified for *ptermname* in each PTERM statement so that the UTM generation matches the terminal generation under the Unix system.

X

X

X

X

Under Unix systems, the default *ptermname* assigned by openUTM may not be unique. Depending on the type of network to which the system is connected, it is possible to have two or more pseudo terminals for which the last term of the `tty` (after the last slash) is identical. Only one terminal can use this *ptermname* to establish a connection with the application. The connection request from the second terminal will be rejected by openUTM.

X

X

X

X

X

X

Example

X

The system contains the `tty`s `/dev/pts/12` and `/dev/inet/12`. If terminal `/dev/pts/12` requests a connection to the application with the *ptermname* 12 and terminal `/dev/inet/12` is already linked to the application, the connection request issued by `/dev/pts/12` is rejected. You must use the last two parts of the output of the `tty` command as the *ptermname*; e.g. instead of PTERM 12, enter the statements PTERM `pts/12` and PTERM `inet/12`.

X

X

X

X

X

X

X

You can also generate an LTERM pool with PTYPE=TTY instead.

W

Terminals connected to a UTM application under Windows systems

W

Any name can be specified for *ptermname*.

BCAMAPPL=local_appliname

Name of the local UTM application as defined in MAX ...,APPLINAME= or the BCAMAPPL statement (see also [page 291](#)). When establishing a connection between the client/printer and the UTM application, *local_appliname* must be specified as the partner name.

In the case of terminals and printers, the name defined in MAX ...,APPLINAME= must be used here.

For PTERMs with PTYPE=SOCKET you must specify a name in *local_appliname* that is generated using BCAMAPPL ... T-PROT=SOCKET.

The BCAMAPPL name specified in the CLUSTER statement is not permitted here.

Default value:

Value in MAX APPLNAME= (primary name of the UTM application). This default value applies if BCAMAPPL= is not specified or contains blanks only.

B/X B/X B/X B/X	CID=printer_id	This applies only for printers assigned to a printer control LTERM. <i>printer_id</i> is used to identify the printers at the printer control LTERM, and can be up to eight characters in length.
B/X B/X		The printer control LTERM is used to manage printers, printer queues and print jobs.
B/X B/X B/X B/X		If the printer is assigned an LTERM partner for which a printer control LTERM has been defined using LTERM ...,CTERM= <i>ltermname2</i> , the printer itself is also assigned to this printer control LTERM. The combination of <i>ltermname2</i> of the printer control LTERM and CID must be unique.
B/X		Default: A CID is not assigned to the printer
	CONNECT=	Specifies whether or not openUTM establishes a connection to the client or printer when starting the application.
B B		– Under BS2000/OSD CONNECT= is only relevant for TS applications, terminals and printers.
X X		– Under Unix systems CONNECT= is only relevant for TS applications and printers.
W W		– Under Windows systems CONNECT= is only relevant for TS applications.
	YES	When starting the application, openUTM automatically attempts to establish a connection. In the case of printers generated with LTERM ...,PLEV > 0, openUTM does not attempt to establish the logical connection until the PLEV value is exceeded. If a connection cannot be established to a printer or TS application, openUTM makes repeated attempts to establish the connection at intervals defined in MAX ...,CONRTIME.
B B B		<i>BS2000/OSD:</i> If a logical connection to a terminal cannot be set up, this can be performed explicitly by the user at a later point in time.
X X X		<i>Unix systems:</i> When starting the application, openUTM automatically creates a printer process for executing print jobs, which is assigned to <i>ptermname</i> .

NO When starting the application, openUTM does not attempt to establish a connection.

CONNECT=NO must be specified for clients with PTYPE=UPIC-R and UPIC-L.

Default: NO

X
X
X
X
X
X



Unix systems:

A printer process is not created for *pservername*. This can only be achieved by issuing the administration command KDCPTERM ACT=C (or KDCLTERM for the assigned LTERM partner). The printer process can then accept print jobs from work processes, which are directed to the appropriate spool queue.

ENCRYPTION-LEVEL=

Only relevant for UPIC clients that support encryption and under BS2000/OSD for some terminal emulations that support encryption also.

In ENCRYPTION-LEVEL you set the minimum encryption level for the communication with the client.

You specify whether or not the UTM application should request encryption of the message on the connection to the client. You can also define the client as a "trusted" client. See also [section "Message encryption on connections to clients" on page 228](#) for more information on encryption.

The client must be a openUTM-Client with the UPIC carrier system and with encryption functions to be able to encrypt data on the connection to the client.

B
B

A prerequisite for the use of encryption on connections between openUTM and terminal emulations is VTSU-B ≥ V12.0C.

You can specify the following:

NONE Encryption of the messages exchanged between the client and the UTM application is **not** requested by openUTM by default. Passwords are transmitted in encrypted form if both partners support encryption. They are transmitted with the longest available key, i.e. with AES if an RSA key with a length of ≥ 512 bits is available; otherwise with DES. The AES key is also encrypted with the longest RSA key available. Services for which encryption was generated for their service TACs (see ENCRYPTION-LEVEL in the TAC statement starting on [page 483](#)) can only be started by this client if the client negotiates encryption when establishing the connection or establishing the conversation.

1 | 2 | 3 | 4 Encryption of the messages exchanged between the client and the UTM application is requested by openUTM by default. The value (1 ... 4) specifies the encryption level. The client cannot connect unless it supports at least this encryption level. Otherwise openUTM rejects connection setup.

Values 1 to 4 have the following meaning:

- 1 Passwords and input/output messages are encrypted using the DES algorithm. An RSA key with a key length of 200 bits is used to exchange the DES key.
- 2 Passwords and input/output messages are encrypted using the AES algorithm. An RSA key with a key length of 512 bits is used to exchange the AES key.
- 3 Passwords and input/output messages are encrypted using the AES algorithm. An RSA key with a key length of 1024 bits is used to exchange the AES key.
- 4 Passwords and input/output messages are encrypted using the AES algorithm. An RSA key with a key length of 2048 bits is used to exchange the AES key.

ENCRYPTION-LEVEL=1 to 4 makes sense for UPIC partners only if the encryption functionality of openUTM is installed on your system. Otherwise the client cannot connect.

B

VTSU encryption is used for VTSU partners.

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 under BS2000/OSD. Levels 2, 3 and 4 are changed to 1 by KDCDEF without issue of a message.
- Encryption level 1, 2, 3 or 4 is changed to TRUSTED without issue of a message for local UPIC clients (PTYPE=UPIC-L) of an application under Unix systems or Windows systems.
- If 1 to 4 is specified for a partner of another type, the value is changed to NONE by openUTM without issue of a message.

B

B

B

X/W

X/W

X/W



If the application is generated with OPTION GEN-RSA-KEYS=NO, no RSA keys are created in the KDCDEF run. In order to use the encryption functions, you must create the required keys using administration facilities (KC_ENCRYPT or WinAdmin) or transfer them from an old KDCFILE using KDCUPD.

TRUSTED

The client is a "trusted" client.

Messages between the client and the application are not encrypted.

A "trusted" client can also start services whose service TACs request encryption (generated with TAC ENCRYPTION-LEVEL=1 | 2).

TRUSTED should only be selected if the client is not accessible for everyone and communication is conducted through a secure connection.

TRUSTED is the only value allowed for local UPIC clients (UPIC-L). Any other specification is changed to TRUSTED by openUTM without announcement.

Default: NONE

X/W
X/W
X/W

IDLETIME=time

May only be specified for dialog partners.

In *time* you enter the maximum time in seconds that openUTM may wait for input from the client outside of a transaction, i.e. after the end of a transaction or after signing on. If this time is exceeded, then openUTM clears down the connection to the client. If the client is a terminal, then message K021 is output before the connection is cleared.

The value specified for *time* may not be smaller than the value of the timer TERMWAIT=*time* (wait time between dialog output and response within a multi-step transaction; see [page 400](#)) and PGWTTIME=*time* (time that a program unit waits for an incoming message after a blocking call; see [page 391](#)) that you specified in the MAX statement.

This function serves to improve data security:

If a user forgets to sign off from the terminal when taking a break or when finishing his or her work on the terminal, then the connection to the terminal or client is automatically cleared down after the wait time has run out. This reduces the chance of someone gaining unauthorized access to the system.

Default: 0 (= no wait time limit)

MAX TERMWAIT=(...,*time2*) is used for terminals (when it is set).

Maximum value: 32767

Minimum value: 60

If you specify a value that is smaller than the minimum value, KDCDEF replaces the value with the minimum value.

LISTENER-PORT=number

Port number for establishing TCP/IP connections

With socket applications (T-PROT=SOCKET) the LISTENER-PORT= is a mandatory operand.

B

BS2000/OSD:

B

Under BS2000/OSD LISTENER-PORT may only be specified for socket applications (PTYPE=SOCKET).

B

In this case, LISTENER-PORT is a mandatory operand.

B

B

For *number* you must specify the port number on which the socket application waits for requests to establish a connection, i.e. the port number on the host partner through which the socket application is addressed.

B

B

B

B

All port numbers are allowed.

X/W

Unix systems and Windows systems:

X/W

LISTENER-PORT is relevant for remote UPIC clients (PTYPE=UPIC-R) and TS applications (PTYPE=SOCKET or APPLI).

X/W

X/W

Permitted values:

X/W

T-PROT=RFC1006: Port numbers 102 and 1025 - 32767

X/W

T-PROT=SOCKET: Port numbers 1 - 65535.

X/W

The LISTENER-PORT is used with T-PROT=SOCKET to specify the port number used to address the partner. No other addressing information is necessary.

X/W

X/W

X/W

X/W

Default: 0 (no port numbers)

X/W

When OPTION CHECK-RFC1006=YES, a port number must be specified in LISTENER-PORT for PTERMs with PTYPE=APPLI or SOCKET.

X/W

LTERM=ltermname

Name of the LTERM partner assigned to the client/printer *ptermname*. This name is used by the client/printer to sign on to the UTM application, and can be up to eight characters in length.

The LTERM operand is mandatory for clients with PTYPE=SOCKET, APPLI and UPIC-R.

The LTERM partner assigned to a client/printer can be changed during runtime using the KDCSWTCH administration command, e.g. if the printer fails. However, it is not possible to assign a dialog LTERM partner (LTERM USAGE=D) to a printer.

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



For the printer pool function, you must issue several PTERM statements with the same *ltermname*. A printer pool consists of numerous printers assigned to a single LTERM partner (see also [section “Generating printer pools” on page 178](#)). openUTM then distributes print jobs cyclically to the various printers in the pool.

MAP=

In MAP you specify whether or not openUTM is to convert the code of user messages exchanged with the communication partner (ASCII <-> EBCDIC).

User messages are passed in the message area on the KDCS interface in the message handling calls (MPUT/MGET/FPUT/DPUT/FGET).

MAP= controls the conversion when exchanging unformatted messages with other applications. openUTM does not generally execute any message handling for formatted messages.

MAP≠USER is only permitted

B
X/W
X/W

- under BS2000/OSD for socket applications (PTYPE=SOCKET).
- under Unix systems and Windows systems for all TS applications (PTYPE=SOCKET or APPLI).

USER

openUTM does not convert the data of the message area, i.e. the messages are transferred to the partner application unchanged and the messages received from the partner are transferred unchanged to the program unit. Note that the user message contains the transaction code in TS applications (partners with PTYPE=SOCKET or APPLI). It must be encoded in the form that the receiving system expects, i.e. under BS2000/OSD in EBCDIC and in ASCII under Unix systems and Windows systems.

Default: USER

B B B B	<p>SYSTEM / SYS / SYS1 / SYS2 / SYS3 / SYS4 (BS2000/OSD)</p> <p>This parameter may only be specified when the messages received by the socket application are not encoded in EBCDIC, or when the socket application expects messages encoded in ASCII from the UTM application.</p>
B B B B B	<p>If you specify one of the values above, then openUTM converts the data in the message from EBCDIC to ASCII before the message is sent and from ASCII to EBCDIC after receiving a message. openUTM assumes that the messages only contain printable characters when converting back and forth.</p>
B B B B B	<p>You specify the conversion table to be used by openUTM for the code conversion with SYSTEM, SYS, SYS1, SYS2, SYS3, SYS4. The conversion tables must be defined in the module KDCEA (see the section "ASCII-EBCDIC code conversion" in the openUTM manual "Programming Applications with KDCS").</p>
B B B B B	<p>SYSTEM, SYS and SYS1 are synonyms. If you specify one of these values, then openUTM uses the standard code table for the conversion that converts EBCDIC to 7-bit ASCII. The standard code table (Table 1) is already defined in KDCEA and can be used without any additional preparation.</p>
B B B B	<p>If openUTM is to execute a different conversion from EBCDIC to ASCII, then you must define the corresponding conversion table yourself in KDCEA. You can define up to a maximum of three conversion tables in KDCEA (Table 2 through Table 4).</p>
B B B	<p>If you specify SYS2, then openUTM converts the user messages using Table 2. openUTM uses Table 3 when SYS3 is specified and Table 4 when SYS4 is specified.</p>
B	<p>This parameter is only permitted for PTYPE=SOCKET.</p>
X/W X/W X/W X/W X/W X/W X/W	<p>SYSTEM (under Unix systems and Windows systems)</p> <p>openUTM converts the data in the KDCS message area from ASCII to EBCDIC before sending messages, or from EBCDIC to ASCII after receiving messages. Messages must contain printable characters only, and must be created in line mode (KCMF = blank).</p> <p>This parameter is permitted only in conjunction with PTYPE=SOCKET or APPLI.</p>

PRONAM={ processorname | C'processorname' }

Symbolic name of the host partner up to 8 characters in length.

If *processorname* contains special characters it must be entered as a character string using C'...'.

B

BS2000/OSD:

B

This name is defined during generation of the network. Please consult your network administrator. The assignment of *ptermname* to *processorname* must be unique.

B

B

B

If a TS application is described with which the UTM application communicates via the socket interface, then you must specify the symbolic address of the host partner for *processorname*. The association of the symbolic address to the real IP address must be entered in the name service of the local system (in the RDF file). You must not specify an alias of the host.

B

B

B

B

B

When defining an RSO printer (PTYPE=*RSO), you must specify *RSO here.

B

B

B

This is a mandatory operand.

B

PRONAM need not be specified if a default value for this operand is defined beforehand using the DEFAULT statement.

B

X/W

Unix systems and Windows systems:

X/W

Under Unix systems and Windows systems PRONAM= is permitted only for remote UPIC clients (PTYPE=UPIC-R) and TS applications (PTYPE=SOCKET or APPLI).

X/W

X/W

X/W

You have two options for specifying the *processorname*:

X/W

- You enter the real host name under which the IP address of the partner computer is entered in the name service of the local system (e.g. the hosts file). You must not specify alias names of the computer.

X/W

X/W

X/W

- You enter the UTM host name of the partner computer. This is only possible when you have set the UTM_NET_HOSTNAME environment variable and specified the UTM host name in the conversion file (see the [section “Using mapped host names \(Unix systems and Window systems\)”](#) on page 122).

X/W

X/W

X/W

X/W

X/W

processorname is a mandatory operand.

X/W

Default: 8 blanks

B B	PROTOCOL=	User services protocol used on connections between the UTM application and the client/printer
B B	N	A user services protocol is not used between the UTM application and the client/printer.
B B B B		PROTOCOL=N must be set for UPIC clients (PTYPE=UPIC-R), TS applications (PTYPE=SOCKET or APPLI) that communicate with the UTM application via the socket interface (native TCP/IP) and for printers accessed via RSO (PTYPE=*RSO).
B B		Clients with PROTOCOL=N cannot sign on to the UTM application via a multiplex connection (MUX statement).
B B		If you specify PTYPE=*ANY, openUTM ignores the entry PROTOCOL=N and automatically sets PROTOCOL=STATION.
B		Default with PTYPE=SOCKET, APPLI, UPIC-R or *RSO.
B B	STATION	The user services protocol (NEABT) is used between the UTM application and the client/printer.
B B B		PROTOCOL=STATION must be specified for clients generated with PTYPE=*ANY. In this case, openUTM requires the user services protocol (NEABT) to determine the device type of the client or printer.
B B B B		For UPIC clients (PTYPE=UPIC-R), TS applications (PTYPE=APPLI or SOCKET) or printers that are addressed via RSO (PTYPE=*RSO), you are only permitted to specify PROTOCOL=N. If you specify PROTOCOL=STATION it will be ignored.
B		Default with PTYPE≠SOCKET , UPIC-R or *RSO.
	PTYPE=	Type of communication partner
B B		This is a mandatory operand under BS2000/OSD.
B		PTYPE under BS2000/OSD:
B B		For PTYPE you must specify the partner type <i>partnertyp</i> of the client or printer, the value *ANY, or the value *RSO for RSO printers.

B
B
B
B

partnertyp Type of communication partner, i.e. type of client or printer. The value specified in *partnertyp* must match that defined during PDN generation. The partner type must be entered either in the PTYPE parameter here, or using a DEFAULT statement. The following partner types are supported:

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

B

Partner	PTYPE	TERMN
DSS 9748	T9748 ²⁾	FE
DSS 9749	T9749	FE
DSS 9750	T9750 ²⁾	FE
DSS 9751	T9751	FE
DSS 9752	T9752	FF
DSS 9753	T9753	FE
DSS 9754	T9754	FI
DSS 9755	T9755 ³⁾	FG
DSS 9756	T9756 ³⁾	FG
DSS 9763	T9763	FH
DSS 9770	T9770	FK
DSS 9770R	T9770R	FK
FHS-DOORS Front End	DSS-FE	FH
DSS 3270 (IBM)	T3270	FL
DSS X28 (TELETYPE)	THCTX28	C5
DSS X28 (VIDEO)	TVDTX28	C6
FHS-DOORS Front End	DSS-FE	FH
Data station PT80	TPT80	C4
9001 printer	T9001	C7
9002 printer	T9002	FA
9003 printer	T9003	F9
9004 printer	T9004	FD
9001-3 printer	T9001-3	CA
9001-893 printer	T9001-893	CB
9011-18 printer	T9011-18	CC
9011-19 printer	T9011-19	CD
9012 printer	T9012	CE
9013 printer	T9013	C9

	Partner	PTYPE	TERMN
B	9021 printer	T9021	CH
B	9022 printer	T9022	CF
B	3287 printer	T3287	CG
B	Intelligent terminal	THOST	A3
B	PDN application	APDN ¹⁾	A2
B	Transport system application that is not a socket application, e.g.: DCAM, PDN, CMX or UTM application.	SOCKET	A7
B		APPLI	A1
B	Socket application	SOCKET	A7
B	UPIC client	UPIC-R	A5

B ¹⁾ A PDN application can also be generated with PTYPE=SOCKET or APPLI. UTM differentiates between applications and stations. APDN is handled as a station and SOCKET or APPLI as an application.

B ²⁾ The PTYPEs T9748 and T9750 refer to the same terminal type.

B ³⁾ The PTYPEs T9755 and T9756 refer to the same terminal type.

B The VTSU version in which the individual terminals are supported can be found in the respective DCAM, FHS and TIAM manuals. If a terminal is not supported by VTSU, openUTM rejects connection requests from this terminal and outputs UTM messages K064 and K107.

B *ANY A PTYPE=*ANY entry generates a VTSU client. The client/printer is incorporated in the configuration without precise information on the device type. During connection setup, openUTM takes the device type from the user services protocol. Only then can it be determined whether or not the partner type is supported.

B The advantage of PTYPE=*ANY is that it allows you to include clients in the configuration without having to know how they are generated in PDN. The configuration is also easier to maintain if, following regeneration in PDN, these clients can still sign on to the application without having to modify the KDCDEF generation.

B If terminal mnemonics (TERMN operand) are not explicitly generated for clients defined with PTYPE=*ANY, the default terminal mnemonic of the partner type is used for connection setup.

B
B
B

*RSO If PTYPE=*RSO, support is provided for printers via RSO. Instead of establishing a transport connection, openUTM reserves the printer in RSO and transfers the message to be printed to RSO.

X/W

PTYPE under Unix systems and Windows systems:

X/W

X/W

partnertyp Type of communication partner, i.e. type of client or printer. For *partnertyp* you can specify the following:

X/W

X/W

X/W

X/W

X/W

X/W

X/W

TTY	The client is a terminal. Default: TTY
APPLI	The client is a transport system application that does not use the socket interface (for example UTM, CMX or DCAM application).
SOCKET	The client is a socket application.
UPIC-L	The client is a local UPIC client.
UPIC-R	The client is a remote UPIC client (usually on another system).

X

X

X

X

X

X

PRINTER (only under Unix systems)
Printer without additional parameters.
To output the data, the printer process (*utmprint*) calls the *utmlp* script. Parameters are also passed to *utmlp* in the call in addition to the data to be printed. *utmlp* then passes the data by default to the lp command (for information on the *utmlp* script see [page 454](#)).

X (PRINTER, printertype,[class]) (only under Unix systems)
 X Printer with extended parameters.
 X To output the data, the printer process (*utmprint*) calls the *utmlp* script.
 X Parameters are also passed to *utmlp* in the call in addition to the data to be
 X printed. *utmlp* passes the data with the value of *class* set to *destination* to the
 X lp command (for information on the *utmlp* script *utmlp* see below).

X *printertype*
 X Designates the printer type of the printer to be used for printing.
 X If there are special characters in the value of the *printertype* parameter, then
 X you must place the value in single quotes.
 X Maximum length of *printertype*: 8 characters

X *class*
 X Name of the printer group (printer class).
 X If the name of the printer group contains special characters, then you must
 X place the value in single quotes.
 X Maximum length: 40 characters
 X Default value: value of *ptermname*

X *Information on the utmlp script*

- X – The *utmlp* script is also supplied. You will find it in the \$UTMPATH/shsc
 X directory.
- X – The parameters are documented in the script itself.
- X – The script is accessed at runtime using the \$PATH variable.
- X – You can edit the script to modify the message before printing or print it
 X over the network, for example.
- X – If printing was successful, the script returns exit code 0 (null).
 X If the script returns an exit code other than 0, then the connection to the
 X printer process is cleared and another attempt is made to output the
 X data once the connection has been reestablished.



With clients of type APPLI, SOCKET or UPIC-R, it may appear to openUTM that a connection to the client still exists, even though the client is no longer actually linked to the application and therefore attempts to reestablish the connection. For this purpose, the client sends a connection request to openUTM, which causes openUTM to shut down the “existing” connection.

With clients of type APPLI or SOCKET, openUTM then automatically initiates the setup of a new connection.

For UPIC clients, the initiation to establish a new connection must be made by the UPIC client.

- STATUS= Status (locked or unlocked) of the client/printer when the application is started.
- ON The client or printer is unlocked. If the LTERM partner used by the client/printer to sign on to the UTM application is not locked, connections can be establish or may already be in place.
 Default: ON
- OFF The client or printer is locked. Connections cannot be established between the client/printer and the local application. The client/printer can be released by the administrator.

TERMN=termn_id

Identifier up to two characters in length, which indicates the type of client. openUTM provides this identifier to the application program in the KCTERMN field of the KB header.

termn_id is not queried by openUTM, but can be used by the user for analysis purposes.

Default values:

If this operand is not specified, openUTM sets the KCTERMN field to the default ID of the partner type specified in the PTYPE operand. However, the user can select other values if desired.

B
B
B
B
B
B
B
B

- *BS2000/OSD:*
 The default values are listed in the partner type table for the PTYPE= operand [on page 451](#).
 If TERMN is not explicitly specified for clients generated with PTYPE=*ANY, openUTM does not enter the terminal mnemonic in KCTERMN until the connection is established. This is the default terminal mnemonic of the type specified in the user services protocol of the connection request.

X/W
X/W

- *Unix systems and Windows systems:*
The default values are listed in the table below.

X/W

X/W

X

X

X/W

X/W

X/W

X/W

PTYPE	TERMN
TTY	F1
PRINTER	F2
PRINTER,fotyp,class	F2
APPLI	A1
UPIC-L	A2
UPIC-R	A5
SOCKET	A7

X/W

T-PROT=

The address format with which the OSI TP partner signs on to the transport system. Is only relevant for PTYPE=SOCKET, APPLI and UPIC-R.

X/W

X/W

X/W

Information on the following address formats can be found in the "CMX User Guide" and in the PCMX online help system.

X/W

RFC1006

Address format RFC1006

X/W

SOCKET

Communication is conducted via the socket interface.

X/W

SOCKET may only be specified if the name of the local UTM application that you specified in BCAMAPPL is generated with T-PROT=SOCKET.

X/W

The specification of a port number in the LISTENER-PORT operand is mandatory.

X/W

X/W

The default value for T-PROT depends on the PTYPE specification:

X/W

T-PROT=RFC1006 when PTYPE=APPLI or UPIC-R

X/W

T-PROT=SOCKET when PTYPE=SOCKET

X/W

TSEL-FORMAT=

X/W

The format identifier of the T-selector in the transport address of the client. TSEL-FORMAT is only relevant for PTYPE=SOCKET, APPLI and UPIC-R.

X/W

X/W

The format identifier specifies the coding of the T-selector in the transport protocol. You will find more information in the "CMX User Guide" and in the PCMX online help system.

X/W

X/W

X/W

T

TRANSDATA format

X/W

E

EBCDIC format

X/W

A

ASCII format

X/W X/W		It is recommended to explicitly specify a value for TSEL-FORMAT for operation via RFC1006.
B B	USAGE=	This specifies whether the communication partner is a dialog partner or purely an output medium.
B B	D	The client is a dialog partner. Messages can be exchanged between the client and the local application in both directions.
B		UPIC clients (PTYPE=UPIC-R) are always dialog partners.
B B		An LTERM partner with USAGE=D must not be assigned to a client with USAGE=O.
B B		This is the default value if PTYPE=SOCKET, APPLI, UPIC-R, and for terminals.
B B	O	The communication partner is a printer. Messages can only be sent from the application to the printer.
B B		Default: This is the only value permitted for partners generated with PTYPE=*RSO.
	USP-HDR=	This parameter is used to control the output messages for which openUTM is to establish a UTM socket protocol header on this connection. A description of the USP header can be found in the openUTM manual "Programming Applications with KDCS".
		This parameter is only relevant for PTERMs with PTYPE=SOCKET.
	NO	openUTM does not create a UTM socket protocol header for any of the output messages. Default.
	MSG	Only when outputting K messages does openUTM create and prefix the message with a UTM socket protocol header.
	ALL	openUTM creates and prefixes all output messages (dialog, asynchronous, K messages) with a UTM socket protocol header.

QUEUE - reserve table entries for temporary messages queues

The QUEUE control statement allows you to specify the number of temporary queues that are permitted to exist in the application at any one time. In the KDCFILE the appropriate number of table entries are reserved for temporary queues. You can also define the default settings for these queues.

Temporary queues are suitable, for example, for communication between two services. These can be created and deleted dynamically during operation using the KDCS calls QCRE and QREL.

The QUEUE statement may only be specified once during a generation run!

For more information about queues and possible applications please refer the openUTM manual "Concepts und Functions".

```

QUEUE_      NUMBER=queue-number
            [ ,QLEV=queue_level_number ]
            [ ,QMODE = { STD | WRAP-AROUND } ]
    
```

NUMBER=queue-number

Specifies the maximum number of temporary queues that are permitted to exist at any one time during an application run.

Minimum value: 1

Maximum value: 500.000

QLEV=queue_level_number

(Queue Level)

Specifies the standard value for the maximum number of messages that may exist at any one time in a temporary message queue.

The maximum number of messages can be defined specifically using the KDCS call QCRE (KCLA parameter) for each queue when the queue is generated. The default value generated with QLEV= is used if the value 0 is entered in the parameter KCLA.

QLEV=32767 means that the number of messages in the queue is not limited by default.

Default: 32767 (or in other words, an unrestricted queue length)

Minimum value: 1

Maximum value: 32767 (or in other words, an unrestricted queue length)

QMODE = (Queue Mode)

Determines the behavior of openUTM in the event that the maximum number of messages saved in a temporary queue has been exceeded and the queue level is thus reached.

The value generated here is used when dynamically creating a temporary queue if no other value is specified in the KDCS call QCRE.

STD openUTM rejects all additional messages for the queue with a negative return code if the queue level has been reached.

Default: STD

WRAP-AROUND

openUTM continues to accept messages for the temporary queue, even if the queue level has already been reached. When writing a message to the queue openUTM deletes the oldest messages in the queue and replaces it with the new one.

REMARK - insert a comment line

The REMARK control statement allows you to insert a comment in the KDCDEF control statements. Comments must not extend beyond one line.

REMARK_	comment
---------	---------

comment Any character string

A comment line can also be created by inserting an asterisk * in column 1.

RESERVE - reserve table locations for UTM objects

If you use the functions of the program interface KDCADMI for dynamic configuration during application operation, or want to add dynamic objects using the WinAdmin administration workstation, then you must use the RESERVE statement to reserve table spaces in the object tables of openUTM at the KDCDEF generation.

Further information on dynamic configuration can be found in [chapter “Changing the configuration of an application dynamically” on page 569](#).

The RESERVE statement can only be issued once for each object type. The following is valid for a RESERVE statement with OBJECT=ALL:

- After RESERVE OBJECT=ALL is specified, it is not possible to enter any additional RESERVE statements.
- Before RESERVE OBJECT=ALL all object-specific RESERVE statements are permitted. These object-specific RESERVE statements take priority.

Due to internal dependencies, the KDCDEF generation tool may reserve more objects than specified in RESERVE statements. The exact number of objects of the reserved entries for an object type is output in a message.

```
RESERVE_      OBJECT={ ALL [ ,CARDS=percent11 ] [ ,PRINCIPALS=percent21 ] |
                CON |
                KSET |
                LSES |
                LTAC |
                LTERM |
                PROGRAM |
                PTERM |
                TAC |
                USER [ ,CARDS=percent11 ] [ ,PRINCIPALS=percent21 ] }
                [ { ,NUMBER=number | ,PERCENT=percent3 } ]
```

B

¹ only permitted under BS2000/OSD

OBJECT= Table locations are reserved for objects of the specified type. These objects can then be entered in the configuration dynamically as required.

ALL [,CARDS=*percent1*] [,PRINCIPALS=*percent2*]

B

(CARDS= and PRINCIPALS= are only permitted under BS2000/OSD)
Table locations can be reserved for objects of type CON, KSET, LSES, LTAC, LTERM, PROGRAM, PTERM, TAC and USER, which are then entered dynamically.

B

With objects of type USER, CARDS=*percent1* means that up to *percent1*% of users entered dynamically can be defined with an ID card.

B

PRINCIPALS=*percent2* means that up to *percent2*% of users entered dynamically can be defined with a Kerberos authentication.

B

B

B

Default for *percent1/percent2*: 0%, i.e. no users can be entered dynamically with an identity card or Kerberos authentication.

B

B

Maximum value for *percent1/percent2*: 100%

CON Table entries are reserved for the transport connections to LU6.1 partner applications, for example, for objects of type CON.

KSET Table entries are reserved for the key sets, for example, for objects of type KSET.

LSES Table entries are reserved for the LU6.1 session names, for example, for objects of type LSES.

LTAC Table entries are reserved for the local service names via which the service programs in partner applications can be started. These are objects of the type LTAC.

LTERM Table locations are reserved for objects of type LTERM

Please note that the following object components must be generated statically and cannot be entered dynamically:

- A client with PTYPE=APPLI cannot be assigned dynamically to an LTERM partner without an autosign USER with the name of a statically generated USER.
- A client with PTYPE=APPLI cannot be assigned dynamically to an LTERM partner without an autosign USER defined with the name of a statically generated user.
- the format handling system when using formats
- the sign-on procedure with #formats.

B

B

PROGRAM

Table locations are reserved for objects of type PROGRAM

Objects of type PROGRAM can only be entered dynamically in applications generated with load modules (BS2000/OSD), shared objects (Unix systems) or DLLs (Windows systems).

Please note that the following object components must be generated statically and cannot be entered dynamically:

- Programming languages (PROGRAM ...,COMP=) must be generated statically using the PROGRAM statement.
- With ILCS-compatible languages (COMP=ILCS), the static generation of an ILCS program is sufficient.
- LOAD-MODULEs in PROGRAM must be generated statically using the LOAD-MODULE statement.
- For applications generated without load modules under BS2000, the PROGRAM names specified when entering a new TAC must be generated statically.
- For applications generated without shared objects/DLLs, the program names specified when entering a TAC must be generated statically.

B

B

B

B

B

B

B

X/W

X/W

PTERM

Table locations are reserved for objects of type PTERM.

For each client with PTYPE=APPLI, SOCKET, UPIC-R or UPIC-L, openUTM implicitly creates a USER. If such clients generated dynamically, this must be taken into consideration in the NUMBER= or PERCENT= operand for OBJECT=USER.

Please note that BCAMAPPL names must be generated statically and cannot be entered dynamically.

TAC

Table locations are reserved for objects of type TAC.

Please note that the following object components must be generated statically and cannot be entered dynamically:

- TAC classes
- If TACs are to be created dynamically for X/Open program units, at least one X/Open TAC must be generated statically.

USER [,CARDS=*percent1*] [,PRINCIPALS=*percent2*]

B

(CARDS= and PRINCIPALS= are only permitted under BS2000/OSD)
Table locations are reserved for objects of type USER.

If user IDs have not been generated for an application, i.e. the generation does not contain any USER statements, table locations cannot be reserved for objects of type USER. This is because KDCDEF already reserves an object of type USER internally for each reserved object of type LTERM. The number of users reserved by KDCDEF in this way is output in a UTM message.

B

B

B

B

Under BS2000/OSD CARDS=*percent1* means that up to *percent1*% of users entered dynamically can be defined with an ID card. PRINCIPALS=*percent2* means that up to *percent2*% of users entered dynamically can be defined with a Kerberos authentication.

B

B

B

Default for *percent1/percent2*: 0%, i.e. no users can be entered dynamically with an identity card or Kerberos authentication.
Maximum value for *percent1/percent2*: 100%

B

B

Please note that the following object components must be generated statically and cannot be entered dynamically:

B

B

- the format handling system when using formats
- the sign-on procedure with #-formats

UTM creates an internal user ID for all TS applications (PTYPE=APPLI/SOCKET) and UPIC clients (PTYPE=UPIC-R). The NUMBER or PERCENT specification must be increased appropriately if these PTERMs are to be entered dynamically.

NUMBER=*number*

Maximum number of objects of the specified type which can be entered dynamically.

If OBJECTS=ALL, up to *number* objects of the types listed in the syntax diagram can be entered dynamically.

- NUMBER=0

The number of objects of the specified type can be increased dynamically to the maximum value, i.e. the maximum number of names that can be generated as specified in [section “Number of names” on page 262](#).

– NUMBER≠0

This reduces the storage space occupied by the UTM application. If the number of objects to be reserved is greater than the maximum number of names that can be generated (see [section “Number of names” on page 262](#)), then this statement has the same effect as NUMBER=0.

Minimum value: 0

Maximum value:

32 000 for LTAC, KSET, TAC and PROGRAM

65 000 for CON, LSES, LTERM, PTERM

500 000 for USER, LTERM, PTERM

The following also apply:

- The sum of the reserved entries for an object type and the number of statically generated names of the associated name classes must not exceed the maximum number of permitted entries for these name classes (see [section “Number of names” on page 262](#)).
- The sum of the reserved CONs and PTERMs must not be greater than 500 000.
- The sum of the reserved LSES and USER must not be greater than 500 000.

At the end of the KDCDEF run, the number of entries reserved for each object type is output with message K502.

PERCENT=*percent3*

Number of objects of the specified type which can be entered dynamically, expressed as a percentage of the total number of objects of this type which have been generated statically.

The advantage of a percentage specification is that the number of objects that can be entered dynamically automatically increases at the same rate as the number of statically generated objects of the respective type in each generation (assuming the RESERVE statements are not modified).

PERCENT=*percent3* has the same effect as the equivalent NUMBER=*number*, i.e. PERCENT=0 has the same effect as NUMBER=0.

PERCENT≠0 reduces the storage space occupied by the UTM application. If the number of objects to be reserved is greater than the maximum number of names that can be generated (see [section “Number of names” on page 262](#)), this statement has the same effect as PERCENT=0.

Default: 10

Minimum value: 0

Maximum value: Number of names that can be generated

RMXA - define a name for an XA (database) connection (Unix systems, Windows systems)

- X/W The XA interface standardized by X/Open allows you to link openUTM to any Resource Manager that supports this interface, e.g. the database systems INFORMIX and Oracle.
- X/W openUTM supports this connection via the XA CAE interface (CAE specification).
- X/W A separate RMXA statement must be issued for each Resource Manager.
- X/W The start parameters for the Resource Manager generally determine the database to which openUTM is linked via the Resource Manager.

```
X/W RMXA_ XASWITCH=name
X/W [ ,DLLIMPORT={ YES | NO } ]
```

- X/W XASWITCH=name
- X/W Name of the *xa_switch_t* structure of the Resource Manager, which is made known to openUTM. The value entered for *name* is predefined in the respective Resource Manager. (Further information can be found in [section "Defining database linking" on page 234.](#))
- X/W This is a mandatory operand.
- X/W DLLIMPORT= Specifies how the *xa_switch_t* structure of the Resource Manager is to be addressed.
- W YES Only permitted in openUTM under Windows systems.
- W The *xa_switch_t* structure is addressed with `dllimport`.
- W You must generate DLLIMPORT=YES to connect to Oracle on Windows systems.
- W NO The *xa_switch_t* structure is addressed using `extern`.
- X/W Default: NO

ROOT - define a name for the ROOT table source

The ROOT control statement must be specified when creating a ROOT table source. It can be omitted if only the KDCFILE is to be created, i.e. neither the operand GEN=ALL nor GEN=ROOTSRC is specified in the OPTION statement. The ROOT statement can only be issued once.

ROOT_	rootname
-------	----------

rootname mandatory operand. You have to specify:

B

BS2000/OSD:

B

CSECT name of the KDCROOT table to be incorporated.

B

When using the ROOT dynamic loading mechanism, this module is loaded during application startup from the library specified in the start parameter *TABLIB=libname*. If this is not the case, the module must be linked statically.

B

B

X/W

Unix systems and Windows systems:

X/W


Name part of the file containing the ROOT table source as a C/C++ program. *rootname* is an alphanumeric name up to eight characters in length. The fully qualified name is *filebase/rootname.c* (Unix systems) or *filebase\rootname.c* (Windows systems).

X/W

X/W

X/W

SATSEL - define SAT logging (BS2000/OSD)

- B The SATSEL control statement allows you to define which events from which UTM event
B class are to be logged using SAT (preselection of the events to be logged). This involves
B specifying the event class to which the events belong, and then restricting the logging
B procedure within each event class by defining whether only successful results or only
B unsuccessful results are to be logged.
- B The SATSEL statement can be issued several times. If an event class is specified in several
B SATSEL statements, the values entered in the first statement determine the logging mode.
- B If SAT logging is to be activated when the application is started, this must be defined during
B generation (MAX ...,SAT=ON).
- B If MAX ...,SAT=OFF is generated, you can use SATSEL to define the events to be logged
B in the generation, even if SAT logging is deactivated. In this case, the SATSEL statements
B are not effective, but SAT logging is predefined. When required, SAT logging can then be
B activated during operation (KDCMSAT administration command).
- B The event logging mode can also be defined using the SATSEL operand in the USER (user-
B specific) and TAC (TAC-specific) statements. If entries are made in various statements, the
B following applies:
- B ● Logging is switched on as soon as it is activated in a statement. The logging mode
B (SUCC, FAIL, or BOTH) is unique.
 - B ● SAT logging can be activated in several statements (SATSEL, USER, and TAC state-
B ments). If different logging modes are specified in the various statements, openUTM
B creates a superset of logging modes by ORing the individual settings. However, there
B is one exception: if an event class is set to OFF in the SATSEL statement, logging is
B deactivated for this event class even if it is activated in the USER or TAC statement.
B Further information on the possible combinations of SAT logging conditions and their
B effect can be found in the openUTM manual “Using openUTM Applications under
B BS2000/OSD”.
 - B ● Each event to be logged (apart from SIGN, CHANGE-PW) is assigned to a USER and
B TAC. Logging of an event can thus be activated using the SATSEL statement (activate
B logging for a particular event) or the SATSEL operand of the USER or TAC statement.
- B  You can find further information about the SAT logging and about possible combi-
B nations of conditions of SAT-loggings and their result in the openUTM manual
B “Using openUTM Applications under BS2000/OSD”.

B	SATSEL	{ BOTH SUCC FAIL NONE OFF }
B		, EVENT=(event1, event2, ...)

B	BOTH	Both successful and unsuccessful events of the class specified in EVENT are logged.
B		
B	SUCC	Only successful events of the class specified in EVENT are logged. SAT logging is also performed as defined in the SATSEL operand of the USER and TAC statements.
B		
B	FAIL	Only unsuccessful events of the class specified in EVENT are logged. SAT logging is also performed as defined in the SATSEL operand of the USER and TAC statements.
B		
B	NONE	Event-specific SAT logging is not performed. SAT logging takes place only if activated for a specific user and/or TAC.
B		
B	OFF	None of the events of the class specified in EVENT are logged, even if SAT logging has been activated in the USER or TAC command. This allows you to exclude events from logging which are not relevant to security (e.g. access to TLS areas), and thus to restrict the quantity of log data.
B		
B	EVENT=(event1, event2, ...)	Event classes to be logged. The following event classes can be selected:
B		
B	SIGN	Events that occur when the user signs on.
B		
B	CHANGE-PW	Events that occur when the user password is changed.
B		
B	START-PU	Events that occur when starting a program unit run, or when accepting a dialog or asynchronous job.
B		
B	END-PU	Events that indicate the end of the program unit run.
B		
B	GSSB	Events that indicate access to a global secondary storage area (GSSB).
B		
B	TLS	Events that indicate access to a terminal-specific long-term storage area (TLS).
B		
B	ULS	Events that indicate access to a user-specific long-term storage area (ULS).
B		
B	ADM-CMD	Events that affect the execution of an administration command issued by direct input or via a program interface.
B		

SESCHA - define session characteristics for distributed processing based on LU6.1

The SESCHA control statement allows you to define session characteristics between the local application and the partner application. The set of session characteristics defined here is stored under a name, which can then be assigned to an LPAP partner using the SESCHA= operand of the LPAP statement (see [page 344](#)).

When generating LU6.1 connections, you must bear in mind the information in [section "Distributed processing via the LU6.1 protocol" on page 76](#).

```

SESCHA_      sescha_name
              [ ,CONNECT={ YES | NO } ]
              [ ,CONTWIN={ YES | NO } ]
              [ ,DPN=destination_process_name ]
              [ ,IDLETIME=pacing_count_time ]
              [ ,PACCNT=pacing_count_number ]
              ,PLU={ YES | NO }

```

X/W *further operand for Unix systems and Windows systems*

```

X/W            [ ,MAP={ USER | SYSTEM } ]

```

sescha_name Name under which the session characteristics are combined. This is specified for the SESCHA= operand of the LPAP statement in order to assign these session characteristics to a particular LPAP partner.

CONNECT= This defines whether the local application is to establish the connection to the partner application during startup.

NO The connection to the partner application must be established using an administration command.

Default: NO

YES The connection to the partner application is established when the local application is started.

If unsuccessful, openUTM repeats its attempt to establish the connection at intervals defined in MAX ...,CONRTIME=.

CONNECT=Y can be specified both in the local application and in the partner application. This means that the connection is established automatically as soon as both applications are available.

CONTWIN= (contention winner)

This defines whether the local application is the contention winner or the contention loser. The contention winner application is responsible for managing the session and controlling the reservation of sessions by jobs. You must specify CONTWIN=Y in one of the two participating applications and CONTWIN=N in the other.

YES The partner application is the contention winner.

NO The local application is the contention winner.

In both cases, jobs can be started by either application. If both applications simultaneously attempt to initiate a job, the session is reserved by the job issued by the contention winner.

The correct selection of this parameter is important for performance in communication between two applications: CONTWIN=Y must be specified in one of the applications, and CONTWIN=N in the other.

Default:

If PLU=N, the local application is the contention loser; otherwise, it acts as the contention winner.

DPN=destination_process_name

Entity that processes asynchronous messages. This operand is significant only for links to IBM systems.

Default: 8 blanks

IDLETIME=dle_time

Number of seconds for which the idle state of a session is monitored. If the session is not reserved by a job within the period specified in IDLETIME=, openUTM shuts down the connection.

IDLETIME = 0 means that the idle state of the connection is not monitored.

Default value: 0

Minimum value: 0

Maximum value: 32767

X/W	MAP=	This controls ASCII/EBCDIC conversion when exchanging unformatted messages with other applications. openUTM does not generally execute any message handling for formatted messages (KCMF contains a format identifier).
X/W		
X/W		
X/W		
X/W	USER	openUTM does not perform message handling, i.e. the data in the KDCS message area is transferred to the partner application unchanged. Default: USER
X/W		
X/W		
X/W	SYSTEM	openUTM converts the data in the KDCS message area from ASCII to EBCDIC before sending messages, or from EBCDIC to ASCII after receiving messages. Messages must contain printable characters only.
X/W		
X/W		

PACCNT=pacing_count_number

Maximum number of message segments of a long message which can be received by the local application without issuing a response. If this value is too high, this may result in network congestion.

If PACCNT=0, pacing does not take place.

Default: 3

Minimum value: 0

Maximum value: 63



CAUTION!

If only short messages are to be exchanged with the partner application (less than 4000 byte) then pacing should be deactivated (PACCNT=0); this saves on overhead in communication with the partner. If data flow problems still occur, then either the default must be reset or the generation of the transport system must be modified accordingly.

PLU=	Application that opens the session, i.e. the primary logical unit (PLU).
YES	The partner application is the primary logical unit.
NO	The local application is the primary logical unit.
	PLU=Y must be specified for one of the applications, and PLU=N for the other.

SFUNC - define function keys

The SFUNC control statement allows you to assign

- transaction codes,
- KDCS return codes transferred to the program units,
- KDC commands, and
- the stacking function

to the function keys of terminals.

It should be issued once for each function key to be used.

A function key can be selected in UPIC clients and transferred to the UTM application. If openUTM receives a function key from a UPIC client, then only the parameter RET is evaluated. If the parameter is not generated, openUTM returns the return code 19Z for the MGET call.

B
B



- F and K keys reserved using SFUNC cannot be used by FHS-DE (see the "FHS User Guide").

X/W
X/W

- Under Unix systems and Windows systems function keys are only relevant for UPIC clients. Only the RET operand is evaluated.

```
SFUNC_      functionkey
            { [ ,CMD={ KDCDISP | KDCFOR1 | KDCLAST | KDCOFF | KDCOFF-BUT |
                    KDCOUT | KDCSIGN1 } ]
            |
            [ ,TAC=tac1 ] { [,RET=xxZ] | [,STACK=tac2] } }
```

B

¹ only permitted under BS2000/OSD

functionkey Short name for the function key. The following values are possible:

B
X/W

BS2000/OSD: K1 to K14 and F1 to F24
 Unix systems and Windows systems: F1 to F20

With F keys, the value of the F key and an input message are displayed.

B
B
B

With K keys, a short message is output indicating the value of the K key. K14 is required for ID card readers (see the openUTM manual "Programming Applications with KDCS").

- CMD=** Name of the KDC command to be assigned to this function key. If the CMD is specified, it is not possible to define any further operands.
- TAC=tac1** Name of the transaction code to be assigned to the function key. This must be defined as a service TAC using the TAC statement (CALL=FIRST/BOTH). If the function key is pressed outside the service, this has the same effect as entering the transaction code. If the function key is pressed within the service and neither RET nor STACK is specified, the first MGET call in the next program unit of this service issues return code 19Z.
- RET=xxZ** Return code contained in the KCRCCC field of the communication area following an MGET call if a particular function key is pressed during a service.
- If the function key at the terminal is pressed at the beginning of a service and TAC=tac1 is not set, openUTM responds by outputting the messaged K009 or by starting the BADTACS program unit. At the first MGET call, the BADTACS program unit receives the return code assigned to the function key in the field KCRCCC.
- If the function key activated from the UPIC client at the beginning of a conversation, that service is started that belongs to the TAC (TP_NAME) set by the UPIC client. At the first MGET call, the program unit receives the return code assigned to the function key in the field KCRCCC.
- The RET and STACK operands are mutually exclusive.
- Value range: $20 \leq xx \leq 39$.
The assignment is freely selectable.
- STACK=tac2** Name of the transaction code to be assigned to the function key. This must be defined as a dialog service TAC using the TAC statement (TYPE=D and CALL=FIRST/BOTH). If the function key is pressed within the service, the current service is stacked and the service with the transaction code tac2 is started. If the function key is pressed outside the service, transaction code tac1 is started. If transaction code tac1 is not specified, pressing the function key starts the service with the transaction code tac2.
- The RET= and STACK= operands are mutually exclusive.

B Alternative assignments under BS2000/OSD

B The following alternative assignments can be specified for K and F keys not available on the
B keyboard:

B	Key	Alternative
B	K1	
B	K2	
B	K3	
B	K4	ESC V
B	K5	ESC W
B	K6	ESC M
B	K7	ESC N
B	K8	ESC O
B	K9	ESC ?
B	K10	ESC >
B	K11	ESC =
B	K12	ESC <
B	K13	ESC ;
B	K14	ESC :
B	F1	
B	F2	
B	F3	
B	F4	ESC ^
B	F5	ESC _


B The 9763 terminal has 24 function keys (F1 to F24). Keys F1 to F20 are activated by
B pressing **SHIFT and** the appropriate Fx(x) key. Keys F21 to F24 are activated by pressing
B **CTRL and** the appropriate Fxx key.

SHARED-OBJECT - define shared objects/DLLs (Unix systems, Windows systems)

- X/W The SHARED-OBJECT control statement allows you to define
- X ● under Unix systems: the name and properties of a shared object if programs are to be exchanged using the dynamic linker.
 - X
 - W ● under Windows systems: the name and properties of DLLs used for dynamic loading.
 - X The program exchange functions are supported on all Unix systems except AIX systems.

```
X/W SHARED-OBJECT_ shared_object_name
X/W          [ ,DIRECTORY=directory_name ]
X/W          [ ,LOAD-MODE={ STARTUP | ONCALL } ]
X/W          [ ,VERSION=version ]
```

- X/W `shared_object_name`
X/W Name of the shared object/DLL up to 32 characters in length.
- X/W `DIRECTORY= directory_name`
X/W Directory in which the shared object is stored. *directory_name* can be up to
X/W 54 characters in length.
X/W Default: No entry, i.e. the current directory is used.
- X *Unix systems:*
X If a directory is not specified for `DIRECTORY=`, the *filebase* directory is
X searched for the shared object. If the shared object cannot be found there,
X the environment variable `LD_LIBRARY_PATH` is used as the search path.
- X/W `LOAD-MODE=`
X/W Load mode of the shared object/DLL
- X/W `STARTUP` The shared object/DLL is loaded when the application is started.
X/W Default: `STARTUP`
- X/W `ONCALL` The shared object/DLL is loaded when the first call of a program unit or of
X/W a conversation exit is issued.
X/W Shared objects/DLLs generated with `LOAD-MODE=ONCALL` can only be
X/W exchanged if the openUTM version support for shared objects/DLLs is
X/W used.

X/W	VERSION=	Shared object/DLL version up to 24 characters in length.
X/W		VERSION= is evaluated only if openUTM version support is used. This is described in the openUTM manual "Using openUTM Applications under Unix systems and Windows systems".
X/W		
X/W		Default: No version specification
W		Under Windows systems it is highly recommended that you use the VERSION= operand since the search for the "lexically largest name" can return unexpected results on Windows systems.
W		
W		
X/W		If VERSION= is not specified and if <i>shared_object_name</i> is a directory name, the shared object/DLL is addressed using the highest version name (in lexical terms). openUTM regards the version name merely as an identifier, i.e. the lexical sequence does not necessary mean "older" or "newer". The UTM administrator is responsible for version management.
X/W		
X/W		
X/W		
X/W		
X/W		Shared object file name without version support:
X		<i>Unix systems:</i>
X		The fully qualified file name of the shared object is
X		<i>directory_name/shared_object_name</i> . If <i>directory_name/shared_object_name</i> is a
X		directory the file is loaded with the highest file name (in lexical terms) from
X		this directory.
X		If the generation statement is
X		SHARED-OBJECT aaa.so, DIRECTORY=.
X		the file ./aaa.so is loaded.
X/W		Shared object file name with version support:
X		<i>Unix systems:</i>
X		The fully qualified file name of the shared object is
X		<i>directory_name/shared_object_name/version</i> .
X		If the generation statement is
X		SHARED-OBJECT aaa, DIRECTORY=., VERSION=V1.S0
X		the file ./aaa/V1.S0 is loaded.
W		<i>Windows systems:</i>
W		The fully qualified file name of the shared object is
W		<i>directory_name\shared_object_name\version</i> .
W		For the generation specification,
W		SHARED-OBJECT aaa, DIRECTORY=., VERSION=V1
W		the file .\aaa\V1.d11 is loaded.
W		The suffix .d11 is added automatically.

SIGNON - control the sign-on procedure

You can specify options and parameters for the sign-on procedure of your UTM application with the SIGNON control statement. The signing on of users is controlled by the SIGNON parameter.

The parameters UPIC, RESTRICTED and CONCURRENT-TERMINAL-SIGNON are only relevant if a sign-on service is generated.

If you enter an invalid value for the SIGNON operand, then KDCDEF uses the corresponding default value. This is currently done without outputting a corresponding message (see the following descriptions of the operands).

```

SIGNON_    [ CONCURRENT-TERMINAL-SIGNON=%_value ]
           [ ,GRACE={ NO | YES } ]
           [ ,MULTI-SIGNON={ YES | NO } ]
           [ ,OMIT-UPIC-SIGNOFF={ YES | NO } ]
           [ ,PW-HISTORY=number ]
           [ ,RESTRICTED={ YES | NO } ]
           [ ,SILENT-ALARM=number1 ]
           [ ,UPIC={ YES | NO } ]

```

CONCURRENT-TERMINAL-SIGNON=%_value

This is only relevant when your application is generated with a sign-on service.

You specify the percentage of users generated for which a sign-on service may be active at the same time in CONCURRENT-TERMINAL-SIGNON. openUTM attempts to allocate the necessary resources according to this specification.

The value *%_value* is based only on sign-on services that are started for terminal users and TS applications.

Default: 25 (%)

Minimum value: 1 (%)

Maximum value: 100 (%)

If you enter a value < 1 or > 100 for *%_value*, KDCDEF sets the default value of 25 % without outputting a message.

GRACE= (Grace-Sign-On)
Specifies if a user may still change his or her password after the password validity period has expired (see USER PROTECT-PW, [page 538](#)).

YES The user can still change his or her password after the password validity period has expired.
The change must be made within the sign-on procedure, before the user is completely 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 <i>Set_Conversation_Security_New_Password</i> 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 systems/ 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. 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.

B
B
B
B
B
B
X/W
X/W
X/W

Note the following particularities after regeneration or change generation:

- If, after regeneration (followed by a KDCUPD run), the password of a user becomes invalid because the complexity requirement has been increased, the user can change his or her password in the sign-on service only (using SIGN CP).
- After regeneration (without a subsequent KDCUPD run), openUTM forces users to change passwords generated with a validity period when they first sign on.

NO The user cannot change his or her password after the validity period has expired. The password may only be changed by an administrator after the validity period has expired.

Default: NO

MULTI-SIGNON=

Specifies if a user may be signed on to the application multiple times under the same user ID simultaneously.



The MULTI-SIGNON operand does not have any effect on the receiving and starting of asynchronous services via OSI TP.

YES The following cases can arise:

- The user ID is generated with USER...,RESTART=NO:
In this case the user may sign on to the application a multiple number of times simultaneously. However, the user may only sign on once through an application. The user can also sign on via UPIC, APPLI, SOCKET and OSI TP connections.
- The user ID is generated with USER...,RESTART=YES:
In this case the user may sign on no more than once to the application, although additional job-receiving services can be active in the application for the user if these services are started via OSI TP connections and the commit functional unit was selected.

NO Every user ID may only be signed on once, and no more than one dialog service can be active at a time for each user.

Default: YES

OMIT-UPIC-SIGNOFF=

Specifies whether a user who has signed on over a UPIC connection remains signed on or not after the conversation has finished.

YES

If a user has signed on over a UPIC connection, they remain signed on after the conversation has finished. This user is only signed off

- if another user is passed in the UPIC protocol before a new UPIC conversation is started over the same UPIC connection,
- or when the connection is cleared.

If no other user is passed in the UPIC protocol, no sign-on service is started before the UPIC conversation is started.

If the application is generated without users, the user ID is never changed for an existing connection. In this case, therefore, a sign-on service is only started where necessary before the first conversation is started after the connection has been established.

Default in UTM cluster applications.

NO

If a user has signed on over a UPIC connection, they are signed off after the conversation has finished.

Default in standalone applications.

PW-HISTORY=number

Specifies if and how many password changes are to be maintained by openUTM in the password history.

If you enter a value > 0 for *number*, then openUTM maintains a password history. *number* is the number of passwords for a user ID that are recorded by openUTM.

If a user changes his or her password and if a maximum period of validity is generated for the password in the USER statement, then the new password must be different from the current password and the last *number* of passwords used by the user.

number=0 means that openUTM will not maintain a password history.

Default: 0

Minimum value: 0

Maximum value: 10

If you specify a value > 10 for PW-HISTORY, then KDCDEF sets it to the maximum value of 10.

The password history only applies to the user; the administrator can change the password irrespective of the history.

RESTRICTED=

Specifies if DB calls and access to global UTM storage is prohibited in the first part of the sign-on service.

YES DB calls and access to global UTM storage is prohibited in the first part of the sign-on service.

NO DB calls and access to global UTM storage is permitted in the first part of the sign-on service.

Default: YES

SILENT-ALARM=number1

Specifies the number of unsuccessful sign on attempts that may occur one after the other via an LTERM partner or a terminal user. A silent alarm (message K094) is triggered when this number is exceeded. The message is output after *number1* unsuccessful sign-on attempts in a row by a user or by a client.

Default: 10

Minimum value: 1

Maximum value: 100

UPIC=

This is only relevant when a sign-on service is generated in your application. With UPIC= you specify in UPIC if the sign-on service is activated when a UPIC client wants to start a conversation.

YES If a sign-on service is generated for the transport system end point (BCAMAPPL) via which the UPIC client has connected to the application, this is started before every UPIC conversation.

NO No sign-on service is started for UPIC clients.

Default: NO

TAC - define the properties of transaction codes and TAC queues

The TAC control statement allows you to define the name and properties of a transaction code and (permanent) message queues of the UTM application.

Transaction codes are the “calling names” for program units of the application. You must always assign a program name (PROGRAM= operand) to a transaction code.

TAC queues are application-wide message queues that exist independently of a program unit. The operand PROGRAM= may not be specified. TAC queues are service-controlled, which means that the program units of the UTM application are responsible for reading messages from queues, openUTM - unlike transaction codes - does not carry out scheduling.

The **dead letter queue** is a TAC queue with the fixed name KDCDLETQ. It is always available for backing up queued messages sent to transaction codes or TAC queues which absolutely could not be processed, i.e. the maximum number of redelivery attempts may have been exceeded. These messages can be read with DGET BF/BN and moved for further processing to other message queues with DADM MV/MA. You cannot generate or process messages for the dead letter queue KDCDLETQ.

The backing up of queued messages in the dead letter queue can be enabled and disabled for each message destination individually using the DEAD-LETTER-Q parameter of the TAC statement. Main jobs to message complexes with negative acknowledgement jobs are never backed up in the dead letter queue.

The name KDCDLETQ is created for the dead letter queue during generation. The following properties are set for the generation:

```
TYPE=Q, STATUS=ON, ADMIN=N, QMODE=STD, QLEV=32767
```

The properties of this TAC queue can also be defined in a separate TAC statement.

A message to a TAC queue cannot be processed when the transaction containing the DGET FT/NT or PF/PN call is reset. A message to an asynchronous TAC cannot be processed when the asynchronous service started with PEND ER/FR terminates abnormally before reaching a synchronization point first.

Generating transaction codes

- The parameters QMODE, Q-READ-ACL and Q-WRITE-ACL have no significance for transaction codes.
- When defining transaction codes for program units containing calls of the X/Open CPI-C or XATMI interface, you must use the API= operand to assign the identifier of the program interface used to the TAC.
- The administration commands used to manage the application must also be defined as TACs. They can be generated as dialog TACs or asynchronous TACs. At least one administration TAC (preferably the KDCSHUT administration command) must be generated and defined in the application. You must also generate at least one user with administration authorization.
- The event services BADTACS, MSGTAC are defined by entering TAC statements with the privileged TAC names KDCBADTC and KDCMSGTC in the generation.
- An event service SIGNON (= sign-on service) may be defined in several ways:
 - using the privileged TAC name KDCSGNTC. You use this to define the event service for the access point specified in MAX APPLINAME=*appliname*. This event service is then also the default for all other access points that are generated using a BCAMAPPL statement.
 - using BCAMAPPL *appliname2*,SIGNON-TAC=*signon-tac* in conjunction with TAC *signon-tac*. You use this to define an own event service for the access point *appliname2*. In this way you can define several SIGNON services.

The event service generated with KDCSGNTC is default for all other access points that are generated with a BCAMAPPL statement.

- For the event services BADTACS, MSGTAC and SIGNON, there are preset values for some operands. These are listed in the table below. These preset values cannot be modified for KDCBADTC, KDCMSGTC and KDCSGNTC. With TAC *signon-tac* that you must set the values as described below.

Operand in TAC statement	Preset value for		
	KDCBADTC	KDCMSGTC	KDCSGNTC or TAC signon-tac
ACCESS-LIST=	Blank	Blank	Blank
ADMIN=	NO	(freely selectable)	NO
API=	KDCS	KDCS	KDCS
CALL=	FIRST	FIRST	BOTH
ENCRYPTION-LEVEL=	NONE	NONE	NONE
LOCK=	0	0	0
SATADM= (BS2000/OSD)	NO	NO	NO
SATSEL= (BS2000/OSD)	NONE	NONE	NONE
STATUS=	OFF	OFF	OFF
TACCLASS=	no TAC class	16	no TAC class
TYPE=	D	A	D

B
BB
B

These default settings mean that, for example, the TACs KDCSGNTC, KDCBADTC and KDCMSGTC are not subject to access protection by key sets and lock codes and cannot be used by the user or specified in a FPUT or DPUT call.

The TACs KDCBADTC, KDCSGNTC and KDCMSGTC are not subject to processing control by the TAC classes. This also applies for KDCMSGTC although KDCMSGTC is assigned to TAC class 16.

All TACs running within a sign-on service are not subject to processing control by TAC classes.

- DEAD-LETTER-Q=NO is set for KDCMSGTC and cannot be changed.

- Note the following when generating TACs:
 - The programs assigned to the TACs KDCBADTC, KDCMSGTC and KDCSGNTC and TAC *signon-tac* must not be assigned to a load module to be loaded dynamically when the first call of one of its program units is issued (LOAD-MODULE statement with LOAD-MODE=ONCALL).
 - The event exit VORGANG and the program units of the service must be located in the same load module if the load module is generated with LOAD-MODE=ONCALL.
- UTM SAT administration commands (preselection commands) can only be generated as dialog TACs. The names of these TACs can be found in the openUTM manual “Using openUTM Applications under BS2000/OSD”.

Generating TAC queues

Only the following operands of TAC statements are relevant for the generation of a TAC queue (TYPE=Q):

tacname, ADMIN, DEAD-LETTER-Q, QLEV, QMODE, Q-READ-ACL, Q-WRITE-ACL, STATUS and TYPE.

The ADMIN, QLEV, QMODE, Q-READ-ACL, and STATUS operands can be used as desired for the dead letter queue KDCDLETQ.

All other operands are not evaluated for TAC queues.



More information about TAC queues and the applications they make possible can be found in the openUTM manual “Concepts und Functions”.

```

TAC_      tacname
          [ , { ACCESS-LIST=keysetname | LOCK=lockcode } ]
          [ , ADMIN={ YES | NO | READ } ]
          [ , API={ KDCS | ( XOPEN, { XATMI | CPIC } ) ]
          [ , CALL={ BOTH | FIRST | NEXT } ]
          [ , DEAD-LETTER-Q={ NO | YES } ]
          [ , ENCRYPTION-LEVEL={ NONE | 1 | 2 } ]
          [ , EXIT=conversation_exit ]
          [ , PGWT={ NO | YES } ] only allowed when TAC-PRIORITIES are used
          [ , PROGRAM=objectname ] only allowed with TYPE=D|A
          [ , QLEV=queue_level_number ]
          [ , QMODE = { STD | WRAP-AROUND } ]
          [ , Q-READ-ACL = keysetname ]
          [ , Q-WRITE-ACL = keysetname ]
          [ , STATUS={ ON | OFF | HALT | KEEP } ]
          [ , TACCLASS=tacclass ]
          [ , TACUNIT=tacunit ]
          [ , TYPE={ D | A | Q } ]

```

B***further operands for BS2000/OSD*****B**

[, DBKEY=dbkey]

B

[, RUNPRIO=priority]

B[, SATADM={ NO | YES }]**B**[, SATSEL={ BOTH | SUCC | FAIL | NONE }]**B**

[, TCBENTRY=name_of_tcbentry_statement]

B

[, TIME={ time1 | (time1,time2) }]

X/W***further operand for Unix systems and Windows systems*****X/W**

[, RTIME=rtime]

tacname Name of the transaction code or the message queue (TAC name) up to eight characters in length.



The specified name must be unique and must not be assigned to any other object in name class 1. See also [section “Uniqueness of names and addresses” on page 265](#).

ACCESS-LIST=*keysetname*

This allows you to define user access authorizations for this transaction code. **ACCESS-LIST=** may not be specified in conjunction with the operand **LOCK=***lockcode*.

Under *keysetname* you must specify the name of a key set. The key set must be defined with a **KSET** statement.

A user is then only able to access the transaction code if the key set of the user (**USER ...**,**KSET=**), the key set of the LTERM partner via which the user is signed on and the key set specified under *keysetname* all contain at least once common key code.

If you specify neither **ACCESS-LIST=***keysetname* nor **LOCK=***lockcode* the transaction code is not protected and any user is able to call the transaction code.

Default: no key set

ADMIN= Authorization required by the user in order to call the transaction code, the TAC-queue or a service containing this transaction code as a follow-up TAC.

YES Meaning for one TAC (**TYPE=A** or **D**):
The TAC can only be called by the administrator or by a user with administration authorization. All functions of the program interface for administration can be used in the associated administration program.

Meaning for a TAC queue (**TYPE=Q**):
Only the administrator or a user with administration authorizations may read messages in this queue/write messages to this queue.

NO Administration authorization is not required for this TAC or this TAC-queue

READ Administration authorization is not required for this TAC or this TAC-queue
Only those functions of the program interface for administration that have read-only access to the application data can be used in the associated administration program (only **KDCADMI** with the operation code **KC_GET_OBJECT**).

- API= Program interface used by the program unit belonging to the transaction code
This is a mandatory operand if you use the X/Open CPI-C or XATMI interface.
- KDCS The program unit is a KDCS program.
Default: KDCS
- (XOPEN,CPIC)
The program unit is a CPI-C program.
- (XOPEN,XATMI)
The program unit is an XATMI program.
- CALL= This specifies whether or not a service is started with the transaction code, i.e. whether the transaction code is the first TAC of a service or a follow-up TAC in a service.
- BOTH The TAC can be used as the first TAC or a follow-up TAC in a service.
Default: BOTH
- FIRST The TAC can only be used as the first TAC in a service.
- NEXT The TAC can only be used as a follow-up TAC in a service. No queued jobs can be generated in this TAC.



CPI-C programs must be generated with CALL=FIRST or BOTH.
XATMI programs must be generated with CALL=FIRST.

B
B

DBKEY=dbkey

This is only relevant if the program unit issues database calls.

B
B
B
B
B
B
B
B
B

dbkey is a name with a maximum length of 8 characters under which the activities of this transaction code are registered with the database system. The format of the key depends on the database system used. The DBKEY is currently only used for UDS databases. As of UDS/SQL V1.2, the DBKEY serves as a special indicator for activity in the UDS monitor ("program-name"). In previous versions of openUTM, this name was also used in the context of permission checking (hence the designation DBKEY). For more information, see the chapter "BPRIVACY" in the UDS/SQL manual "Creation and Restructuring".

B

Default: UTM

B
B
B

The default value DBKEY=UTM causes the value of the start parameter DBKEY to be passed at the database interface (see openUTM manual "Using openUTM Applications", Start parameters).

DEAD-LETTER-Q=

Specifies whether asynchronous messages of this message queue are to be placed in the dead letter queue after incorrect processing and unsuccessful redelivery.

The statement `MAX ...,DEAD-LETTER-Q-ALARM` can be used to enable monitoring the number of messages in the dead letter queue.

YES 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 from.

NO Messages to this asynchronous TAC or this TAC queue which could not be processed are not saved in the dead letter queue.

This value must be generated for all dialog TACs, for asynchronous TACs with `CALL=NEXT` and for `KDCMSGTC` and `KDCDLETQ`.

Default: NO



Main jobs to message complexes (MCOM) with negative acknowledgement jobs are never saved into the dead letter queue since the negative acknowledgement jobs are activated if an error occurs.

If the number of messages in the dead letter queue is limited with `QLEV`, messages from asynchronous TACs or TAC queues can be lost if an error occurs. If this limit is not applied, the `openUTM` page pool must be dimensioned large enough. If there is a danger of a page pool bottleneck, the dead letter queue can be blocked during operation with `STATUS=OFF`.

ENCRYPTION-LEVEL=

In `ENCRYPTION-LEVEL` you set the minimum encryption level that must be used by a service started through this transaction code. The encryption level specified here applies to all messages that are sent and received in the service.

NONE Encryption of the messages is not necessary.
You must set `ENCRYPTION-LEVEL=NONE` for transaction codes generated with `CALL=NEXT`.

Default: NONE

1 | 2 A service can only be started with this transaction code if the input message from the client is transmitted in encrypted form.
Dialog output messages of the service are transmitted to the client in encrypted form.

The value (1 or 2) specifies the algorithm to be used for encryption:

- 1 Encryption of input/output messages using the DES algorithm.
- 2 Encryption of input/output messages using the AES algorithm.

If a client does not encrypt the first input message with at least the requisite encryption level or does not support encryption, then no service is started. The following exceptions apply:

- The calling client is generated as a trusted client (PTERM/TPOOL ..., ENCRYPTION-LEVEL=TRUSTED).
- The service is an asynchronous service and is started locally.
- The service is started by means of service chaining.
- The service is started without user data.

If the transaction code is started through service chaining, then the first input message of the client does not have to be encrypted.

If the transaction code is called without user data or started through service chaining, then the client must be able to encrypt because openUTM transmits all dialog output messages in encrypted form and, for multi-step services, expects all additional input messages from non-trusted clients to be encrypted.

You may only specify ENCRYPTION-LEVEL=1 | 2 for transaction codes used to start a service (CALL=FIRST or CALL=BOTH).



Transaction codes with ENCRYPTION-LEVEL=1 | 2 can only be started by clients that are generated as trusted clients in systems in which the openUTM encryption functionality is not installed.

EXIT=conversation_exit Name of the event exit VORGANG to be assigned to this TAC. EXIT= can only be specified in conjunction with CALL=FIRST or CALL=BOTH. The event exit VORGANG must be defined in a separate PROGRAM statement.

Default: No event exit VORGANG

LOCK=lockcode

Lock code assigned to the transaction code of a service in the form of a numerical lock. *lockcode* is a number between 1 and the maximum value permitted by the application (MAX ..., KEYVALUE=).

This may not be specified in conjunction with the operand ACCESS-LIST=.

For data access control, key sets can be defined for (UTM) user IDs (USER) and for the LTERM/(OSI-)LPAP partners. If a service is secured by means of a lock code, it can only be started if the appropriate key code is contained both in the key set of the user ID, **and** in the key set of the LTERM/(OSI-)LPAP partner.

Services whose TACs are not secured with a lock code or an ACCESS-LIST can be called by any user ID and any LTERM/(OSI-)LPAP partner without restriction. Further information on the lock/key code and access list concepts can be found in the openUTM manual “Concepts und Functions”.



CAUTION!

If the user and the LTERM/(OSI-)LPAP partner do not also have the key code for a continuation program called by this TAC, openUTM aborts the service with an error.

Default: 0 (the TAC is not secured with a lock code)

Maximum value: Value of MAX ...,KEYVALUE=*number*

PGWT

You may only specify PGWT if the jobs to TAC classes are processed according to their priority in your application, i.e. the KDCDEF generation contains the TAC-PRIORITIES statement. You specify whether or not blocking calls (e.g. PGWT) are allowed to be executed in a program unit run that was started for this transaction code with PGWT.

YES

Blocking calls are permitted.

If you specify PGWT=YES, then you must assign a TAC class to this transaction code, i.e. you must set TACCLASS= .

Note the following cases:

– CPI-C program units

If a CPI-C program unit is to conduct dialog conversations in which send authorization is transferred to the conversation partner using a *Set_Send_Type* call with *send_type*=CM_SEND_AND_PREP_TO_RECEIVE or by issuing a Receive call in Send status, then the transaction code of this CPI-C program unit must be assigned to a TAC class generated with PGWT=YES.

– XATMI program units

If an XATMI application contains both requests and conversational services, at least two tasks must be started and the transaction code for the service must be generated with PGWT=YES.

NO

Blocking calls are not permitted.

Default: NO

PROGRAM=objectname

Name of the program unit to which this TAC is to be assigned.

A program name must be generated for asynchronous and dialog TACs; the PROGRAM parameter is not permitted for TAC queues.

Default: Blanks, no program name

If the program is not loaded in application operation, or the access authorizations do not permit the call, openUTM calls the BADTACS dialog service. If BADTACS is not generated in the application, UTM outputs the message K009 instead.

QLEV=queue_level_number

(queue level)

For asynchronous transaction codes (TYPE=A), this operand specifies the maximum number of asynchronous messages that can be accommodated in the message queue of the transaction code. QLEV can be used to prevent the page pool from becoming overloaded with jobs for this TAC or this TAC queue. openUTM does not take the asynchronous jobs into consideration until the end of the transaction. It is possible to exceed the number of messages for a messages queue as specified in QLEV if several messages are created for the same queue in a transaction.

If an additional message is to be created once QLEV has been reached, the behavior of openUTM will depend on the setting made in QMODE= (see below).

Default: 32767

Minimum value: 0

Maximum value: 32767 (i.e. unlimited)

If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.

QMODE = **(Queue Mode)**

This determines the behavior of openUTM in the event that the maximum permitted number of messages saved in a queue has already been reached and thus the Queue Level has been reached.

STD If, at the time of an FPUT or DPUT call, the number of messages saved in this queue is greater than or equal to the maximum number generated in QLEV=, the FPUT or DPUT call is rejected with 40Z or with an appropriate message, if this TAC was entered at a terminal.

WRAP-AROUND

Only for TACs with TYPE=Q (TAC queues):

openUTM continues to accept messages for this queue, even when the Queue Level has been reached. When writing the next messages to the queue openUTM deletes the oldest existing message from the queue providing that its start time has been reached and it is not currently being read.

Default: STD

Q-READ-ACL=read-keysetname

This parameter is only evaluated for TACs with TYPE=Q (TAC queues). This parameter is used to specify the authorizations that a user requires to be able to read and delete messages from this queue.

In this parameter you can specify the name of a KSET that is defined with a KSET statement. In this case, a user can only then have read access to this TAC queue if the key set (KSET) of the user and that of the logical terminal via which the user has signed on, both contain at least one key code that corresponds to the key code specified in the key set entered here.

If no key set is specified in Q-READ-ACL, all users are able to read and delete messages from this queue.

Default: no key set

Q-WRITE-ACL=write-keysetname

This parameter is only evaluated for TACs with TYPE=Q (TAC queues). It may not be specified for the dead letter queue.

This parameter is used to set the authorizations that a user requires to be able to write messages to this queue.

Using this parameter you can specify the name of a KSET that has been defined using a KSET statement. In this case, a user can only have write access to this TAC queue if the key set (KSET) of the user and that of the logical terminal via which the user is signed on, both contain at least one of the key codes contained in the key set specified here.

If no key set is specified in Q-WRITE-ACL, all users are able to write messages to this queue.

Default: no key set

X/W

RTIME=rtime

X/W

X/W

Maximum real time (in seconds) available to a program unit started using this TAC. If the program unit runs over the specified time, openUTM terminates the service and outputs an error message (see the openUTM manual "Messages, Debugging and Diagnostics under Unix systems and Windows systems").

X/W

X/W

rtime = 0 means that the program unit real time is not monitored.

X/W

Default: 0

X/W

Minimum value: 0

X/W

Maximum value: 32767

B B B B B B	RUNPRIO=priority	BS2000 run priority of the TAC. This run priority is assigned to the UTM process in which the program unit runs (PROGRAM). You can thus use the BS2000 scheduling mechanisms to control the sequence of UTM program units. However, the RUNPRIO operand cannot influence the time at which openUTM starts a program unit.
B B B B B B B		When starting a program unit, openUTM attempts to set the run priority of the current process to the value defined in RUNPRIO for the current TAC. If the generated run priority is incompatible with the JOIN entries of the corresponding user ID, the run priority of the current process is not changed and openUTM outputs a corresponding K message. If the maximum permitted RUNPRIO values for the user ID and the job class are different, the value most beneficial to the user is permitted. If JOIN entries have not been defined, the run priority specified in RUNPRIO is set.
B B B B		After the program unit is terminated, openUTM resets the run priority to its original value, unless it was changed during the program unit run using the CHANGE-TASK-PRIORITY command. In this case, the run priority set externally is retained after the end of the program unit.
B		If RUNPRIO=0, a TAC-specific run priority is not generated for this TAC.
B B B		Default: 0 Minimum value: 30 (highest priority) Maximum value: 255 (lowest priority)
B B	SATADM=	This defines whether UTM SAT administration authorization is required in order to call the TAC.
B B B	YES	The TAC can only be called by users/clients or partner applications for which administration authorization for SAT logging (PERMIT=SATADM) has been generated in the USER, LPAP or OSI-LPAP statement.
B B	NO	The user/client or partner application does not require UTM SAT administration authorization to use the TAC.
B	SATSEL=	SAT logging mode when running the program unit called using this TAC.
B B		If SAT logging is activated (MAX ...,SAT=ON), TAC-specific events are logged as defined in this operand during a program run under this TAC.
B B B B B B B		The SATSEL control statement is used to define the general SAT logging mode for all TACs and users. This can be supplemented by the SATSEL operand of the TAC statement, which allows you to define TAC-specific logging. If the logging of an event class is prohibited in the SATSEL statement, events of this class are not logged. (For information on the link between EVENT-, TAC- and USER-specific log settings, see openUTM manual "Using openUTM Applications under BS2000/OSD").

B		SATSEL can be generated even if SAT logging is deactivated (MAX ...,SAT=OFF). In this case, the statements are not effective when the application is started, but SAT logging is predefined. When required, SAT logging can then be activated during operation with the UTM SAT administration command KDCMSAT.
B		
B		
B		
B		
B	BOTH	Both successful and unsuccessful events are logged.
B	SUCC	Only successful events are logged.
B	FAIL	Only unsuccessful events are logged.
B	NONE	A TAC-specific SAT logging mode is not defined.
B		Default: NONE
	STATUS=	Status (locked or unlocked) of the TAC or the TAC queue when the application is started.
	ON	<p>Meaning for TACs: The TAC is unlocked, and is available once the application is started, until such time as the administrator locks it.</p> <p>Meaning for TAC queues: Read and write access is permitted for this queue.</p> <p>Default: ON</p>
	OFF	<p>The TAC is locked when the application is started. Jobs for this TAC are not accepted until the TAC is unlocked by the administrator.</p> <p>If the transaction code belongs to a KDCS program unit and is generated with CALL=BOTH or CALL=NEXT, it is locked as a service TAC (first TAC of a service) but not as a follow-up TAC.</p> <p>Meaning for TAC queues: The queue is locked to write access. Read access is permitted.</p>
	HALT	<p>The TAC is locked in full when the application is started, i.e. even as a follow-up TAC in an asynchronous service or a dialog service.</p> <p>If the TAC is called as a follow-up TAC, the service is terminated with PENDER (74Z). The TAC must be released by the system administrator. Asynchronous jobs already buffered in the message queue of the TAC are not started. They remain in the message queue until the TAC status is set to ON or OFF by the UTM administrator.</p> <p>Meaning for TAC queues: The queue is locked to both read and write access.</p>

KEEP May only be specified for TAC queues and for asynchronous transaction codes that are also service TACs (CALL=BOTH or CALL=FIRST). openUTM accepts jobs for the transaction code. The jobs are not processed, however, rather just written in the message queue of the transaction code. They are processed as soon as the administrator changes the status of the transaction code to ON or OFF.

You can use STATUS=KEEP to collect jobs that are to be executed later at a time when the application load is lower (e.g. at night).

To avoid overloading the page pool with too many temporarily stored jobs, you should limit the size of the job queue of the transaction code using the QLEV parameter.

Meaning for TAC queues: The queue is locked to read access. Write access is permitted.



The status is always set to ON for the KDCSHUT and KDCTAC administration commands, even if you specify a different value for STATUS. Your application can always be administered in this manner.

TACCLASS= Assigns the transaction code a TAC class.

The TAC classes are required for controlling the processing of dialog and asynchronous jobs. Jobs that are assigned different TAC classes are started according to different criteria by openUTM. The TAC class, which is assigned a transaction code, controls whether a job is processed immediately or temporarily stored in the message queue of the transaction code first, and when it will be read out of the message queue and processed. There are two different methods available to control job processing (see [section "Job control - priorities and process limitations" on page 208](#)).

tacclass The following numerical values are permitted:

- 1 - 8 for dialog TACs
- 9 - 16 for asynchronous TACs

If asynchronous TAC classes are generated, then the value in MAX ...,ASYNTASKS must be greater than 0.

If your application is generated **without** TAC-PRIORITIES statements and encounters blocking calls in the program unit belonging to this TAC (e.g. the KDCS call PGWT), then you must specify the dialog or asynchronous TAC class for *tacclass* for which **TACCLASS PGWT=YES** is set.

If your application is generated **with** TAC-PRIORITIES statements, then you can assign any dialog or asynchronous TAC class to this TAC. You just need to set **TAC ...,PGWT=YES** in this case.

Default for dialog TACs:

Dialog TACs are normally not assigned to a TAC class. The program unit belonging to the dialog TAC is started as soon as a process retrieves the corresponding message from the job bourse of the application.

Default for asynchronous TACs:

The default value for asynchronous TACs is 16.



If the transaction code is generated with PGWT=YES, then you must assign a TAC class to the transaction code.

TACUNIT= tacunit

Specifies the number of accounting units that are charged in the accounting phase of the UTM accounting each time this transaction code is called. The accounting units are added to the accounting unit counter of the user ID that called the transaction code.

This operand is required only if openUTM is to collect accounting data (see also the ACCOUNT statement on [page 283](#) and "Accounting" in the openUTM manual "Using openUTM Applications". You must enter an integer here.

Default value: 1

Minimum value: 0

Maximum value: 4095

B TCBENTRY=name_of_tcbentry_statement

B Only relevant for transaction codes from program units that are generated
B with PROGRAM ...,COMP=COB1.

B *name_of_tcbentry_statement* designates the name of a TCBENTRY statement
B in which the TCB entries assigned to this TAC have been combined.

B Default: No name

B TIME= Check CPU resource consumption for a program unit.

B time1 Maximum CPU time (in milliseconds) available to the program unit with this
B TAC. If the program unit runs over the specified time, openUTM terminates
B the service and outputs UTM message K017 for dialog programs or K055
B for asynchronous programs. KCRCCC is set to 70Z, and KCRCDC to XT20
B (see the openUTM manual "Messages, Debugging and Diagnostics under
B BS2000/OSD").

B
B
B

The value 0 means that the program unit started using this TAC is not subject to a timeout. Values 1 to 999 are not permitted and are replaced with 1000.

B
B
B

Default: 30000 ms
Minimum value: 0 ms
Maximum value: 86400000 ms

B
B
B
B**CAUTION!**

With the administration TACs KDCSHUT, KDCSHUTA, KDCDIAG and KDCDIAGA, the value of TIME=*time1* should be set to a value greater than the default value (at least twice as large; ≥ 60000 ms).

B
B
B
B
B

With KDCSHUT WARN, applications with large numbers of generated terminals may require more CPU time than permitted by the default value. (See also the openUTM manual “Administering Applications”.) The same is true when you request a diagnostics dump with KDCDIAG DUMP=YES in large applications.

B
B
B
B
B
B
B

time2

Maximum real time (in seconds) available to the program unit with this TAC. If the program unit runs over the specified time, openUTM terminates the service with UTM message K017 for dialog programs or K055 for asynchronous programs. KCRCCC is set to 70Z, and KCRCDC to XTA0 (see the openUTM manual “Messages, Debugging and Diagnostics under BS2000/OSD”). The value 0 means that the real time is not monitored for the program unit started using this TAC.

B
B
B

Default: 0 s
Minimum value: 0 s
Maximum value: 32767 s

TYPE=

This defines whether jobs with this transaction code are processed in dialog, in an asynchronous mode or whether a TAC queue is created.

D

The TAC is a dialog transaction code, i.e. a job with this TAC is processed in the dialog with the job submitter.

Default: D

A

The TAC is an asynchronous transaction code, i.e. a job with this TAC creates an asynchronous job in the message queue of the transaction code. Processing takes place independently of the job submitter.

Q

This TAC statement is used to generate a TAC queue. In a queue of this nature it is possible to use a FPUT or DPUT call to write a message to a queue and to use a DGET call to read a messages in the queue.

TACCLASS - define the number of processes for a TAC class

You specify the method used to control job processing in this UTM application with the TACCLASS control statement. This means that you specify the criteria used by openUTM to start the jobs for transaction codes that have been assigned a TAC class.

The specification of these criteria can also be done using the TACCLASS statement or the TAC-PRIORITIES statement.

A TAC class consists of a subset of the generated transaction codes of the application. These TACs are divided into TAC classes using the TACCLASS= operand of the TAC statement.

By generating at least one TACCLASS statement, you specify that job processing in your application is controlled by the limitation of the number of processes for the individual TAC classes. You may not issue any TAC-PRIORITIES statements in this case, then.

The TACCLASS statement allows you to define how many processes of the UTM application are allowed to work at the same time for the TACs of a TAC class. You can also specify in the PGWT operand whether or not blocking calls (e.g. the KDCS call PGWT) are allowed or not in program unit runs that are started by transaction codes of the TAC class. You may only assign the PGWT=YES property, i.e. blocking calls are allowed, to one dialog and one asynchronous TAC class.

The number of processes of a TAC class that you specify in the TACCLASS statement can be changed by the administrator (see the openUTM manual "Administering Applications").

You can thus control the load on the UTM application exerted by the program units of individual TACs. For example, you can prevent long-running program units from blocking the application. If asynchronous services are used for distributed processing, then you can avoid situations where all the application processes available for asynchronous processing are allocated by this service.

Default values

All TAC classes are created implicitly in the KDCDEF generation if you generate a transaction code with the TAC ...,TACCLASS= statement, or if you generate a TAC class with TACCLASS.

If you do **not** issue any TAC-PRIORITIES statements, then you should write a TACCLASS statement for every TAC class used. In this case, openUTM assigns the minimum value for TASKS and TASKS-FREE to those TAC classes for which no TACCLASS statement is issued. You must always issue TACCLASS statements for TAC classes with PGWT=YES!

If you do not use TAC classes, i.e. the TACCLASS= operand is not specified in any TAC statement and there are no TACCLASS or TAC-PRIORITIES statements, then the following applies:

- Dialog TACs are processed without restriction.
- Asynchronous TACs are restricted by the number of processes specified in the ASYNTASKS start parameter. This value can be modified by the administration.



You will find a detailed description of the TAC classes and priority control in [section “Job control - priorities and process limitations” on page 208](#).

```
TACCLASS_   tacclass
             ,{ TASKS=number1 | TASKS-FREE=number2 }
             [ ,PGWT={ NO | YES } ]
```

tacclass Number of the TAC class for which the number of processes is to be specified. You may specify the following TAC classes:

- the dialog TAC classes 1 - 8
- the asynchronous TAC classes 9 - 16.

You assign a transaction code to this TAC class by specifying TACCLASS=*tacclass* in the corresponding TAC statement.

You may only specify an asynchronous TAC class if you have generated a non-zero value in MAX ...,ASYNTASKS.

Asynchronous transaction codes that are not assigned a TAC class are automatically assigned TAC class 16.



The TAC class numbers are not priorities, rather only a designation for the TAC class. Only the number *number1* of permitted processes can determine the extent to which processing of a TAC class is restricted so that the TACs of other classes can be processed quicker. This is possible only if *number1* is less than the number of active processes of the application.

TASKS=number1

Maximum number of application processes that can be executed simultaneously for the TACs of this class. The values permitted here depend on the value of the PGWT operand and the values defined for the TASKS, TASKS-IN-PGWT, and ASYNTASKS operands of the MAX statement.

The value ranges permitted for TASKS=number1 are given in the table below.

	Class 1 - 8 Dialog TACs		Class 9 - 16 Asynchronous TACs	
Minimum value	PGWT =NO	PGWT=YES	PGWT= NO	PGWT=YES
	1	1	0	0
Maximum value	TASKS *)	TASKS-IN-PGWT *)	ASYNTASKS*)	The lesser of the two values: ASYNTASKS, *) TASKS-IN-PGWT *)

*) As specified in the MAX statement

This is a mandatory operand if TASKS-FREE= is not specified.

If you enter TASKS=0 for a dialog TAC class, openUTM automatically resets this value to 1.



The total number of tasks entered in the TASKS= operand of the individual TACCLASS statements can be greater than the maximum number of processes permitted by the application (MAX ...,TASKS=).

TASKS-FREE=number2

TASKS-FREE specifies for

- dialog TAC classes:
Minimum number of processes of the UTM application to be kept free for processing TACs of other classes.
- asynchronous TAC classes:
Minimum number of processes permitted for asynchronous jobs (MAX ...,ASYNTASKS=) to be kept free for processing TACs of other classes.

Compared to the TASKS parameter, TASKS-FREE offers the advantage of dynamically adapting the number of processes permitted for a TAC class if the total number of application processes is changed.

The values permitted for TASKS-FREE=*number2* depends on the values defined for the TASKS and ASYNTASKS operands of the MAX statement. The value ranges permitted for TASKS-FREE= are given in the table below:

	Class 1 - 8 Dialog TACs	Class 9 - 16 Async. TACs
Min. value	1	1
Max. value	TASKS-1 *)	ASYNTASKS *)

*) As specified in the MAX statement

This is a mandatory operand if TASKS= is not specified.

If you enter TASKS-FREE=0, openUTM automatically resets this value to 1.

PGWT=

(program wait)

This specifies whether or not program units containing blocking calls (e.g. KDCS call PGWT) can be executed in this TAC class.

(Further information on PGWT can be found in the openUTM manual “Programming Applications with KDCS” and in the openUTM manual “Concepts und Functions”).

YES

Blocking calls are permitted in this TAC class.

PGWT=YES can only be generated if MAX ...,TASKS-IN-PGWT ≠0 is defined. It can be specified for up to one dialog TAC class and one asynchronous TAC class. Program units containing PGWT calls must be assigned to this TAC class.

– CPI-C program units

If a CPI-C program unit is to conduct dialog conversations in which send authorization is transferred to the conversation partner using a Set_Send_Type call with

send_type=CM_SEND_AND_PREP_TO_RECEIVE or by issuing a Receive call in Send status, the transaction code of this CPI-C program unit must be assigned to a TAC class generated with PGWT=YES, e.g.:

```
MAX TASKS=2
MAX TASKS-IN-PGWT=1
TACCLASS 1 ,TASKS=1 ,PGWT=YES
TAC CPIC1 ,PROGRAM=xyz ,API=(XOPEN ,CPIC) ,TACCLASS=1
```

– XATMI program units

If an XATMI application contains both requests and conversational services, at least two tasks must be started and a TAC class that permits PGWT calls must be generated. A service is always linked to the task. This is not necessary for applications that only contain request/response services.

NO Program units that contain blocking calls are not permitted in this TAC class.
Default: NO



Blocking calls for asynchronous TACs are not processed until all jobs in the messages queues of dialog TAC classes have been executed.

Example

The table below shows how the value defined for TASKS-FREE affects that defined for the TAC class under certain marginal conditions.

- The CURRENT TASKS column contains the maximum number of processes currently available to the UTM application. CURRENT ASYNTASKS contains the maximum number of processes available for performing asynchronous services. The global maximum values for CURRENT TASKS and CURRENT ASYNTASKS are defined in the TASKS and ASYNTASKS operands of the MAX statement. During runtime, the current values can be modified dynamically within this upper limit using the TASKS and MAXASYN operands of the KDCAPPL command.
- The DIALOG column contains the maximum number of processes available for a particular DIALOG-TAC class if TASKS-FREE=*nn* is specified for this TAC class.
- The ASYNCH column contains the maximum number of processes available for a particular asynchronous TAC class if TASKS-FREE=*number2* is specified for this TAC class.

CURRENT TASKS	CURRENT ASYNTASKS	TASKS-FREE	DIALOG	ASYNCH
10	9	2	8	7
6	6	2	4	4
3	3	2	1	1
2	2	2	1	0
1	1	2	1	0
10	5	3	7	2
6	5	3	3	2

TAC-PRIORITIES - specify priorities of the TAC classes

With the TAC-PRIORITIES control statement you specify the method to be used to control job processing in this UTM application. This means that you specify the criteria used to start jobs for transaction codes that are assigned a TAC class.

You can also specify these criteria using the TAC-PRIORITIES statement or the TACCLASS statement.

A TAC class consists of a subset of the generated transaction codes of the application. The dividing of transaction codes into TAC classes is done in the TAC statement with the TACCLASS= operand.

If the TACCLASS operand is not specified, then dialog TACs are not assigned a TAC class and asynchronous TACs are not assigned asynchronous TAC class 16.

You can specify the following in particular with TAC-PRIORITIES:

- That the distribution of processes amongst the TAC classes is to be done according to priorities. You must not issue any TACCLASS statements in this case.
- The algorithm to be used to distribute the available processes of the application amongst the dialog and asynchronous TAC classes.
Operands: DIAL-PRIO and ASYN-PRIO
- The maximum number of processes of the application that are allowed to process jobs to dialog TAC classes.
Operand: FREE-DIAL-TASKS

Specifying priorities for the TAC classes

In priority control you can select between absolute, relative or equal priorities for dialog jobs and for asynchronous jobs. The control of job processing of dialog and asynchronous jobs is done separately from each other.

Jobs for dialog TACs that are not assigned any TAC class are processed regardless of the priorities set for dialog jobs. These jobs are always started immediately after they are received from the transport system.

The number of the TAC class plays a role for absolute and relative priorities. Jobs to TAC classes with a low number have a higher priority than jobs to TAC classes with a higher number. This means that for dialog TAC classes, TAC class 1 has the highest priority and TAC class 8 the lowest priority. For asynchronous TAC classes, TAC class 9 has the highest priority and TAC class 16 the lowest priority.

When absolute priorities are used, processes of the application that are free and available for processing the TAC classes are always assigned the TAC class with the highest priority, i.e. 1 or 9, as long as there are jobs waiting for this TAC class.

Only after there are no more jobs waiting in the TAC class with the highest priority are jobs waiting for the TAC class with the next lowest priority processed.

If you want to prevent jobs waiting for a TAC class with a lower priority from not being processed for a long time, then you should use relative priorities.

When relative priorities are used, jobs from TAC classes with higher priority are processed more often than jobs from TAC classes with lower priority.

When matching priorities are used, then the same number of jobs from each TAC class are processed as long as there are waiting jobs available.

Limiting the number of processes that process jobs to TAC classes

You can limit the number of processes that process jobs of a TAC class when using priority control for the TAC classes to keep some processes free for administrative tasks or internal jobs.

You limit the number of processes for the dialog TAC classes relative to the total number of processes using the FREE-DIAL-TASKS operand.

With MAX ASYNTASKS=(*atask_number*,...) you limit the number of processes for asynchronous TAC classes.

This limit is the same, however, for all asynchronous and for all dialog TAC classes.

Transaction codes that start program unit runs with blocking calls

When the TAC-PRIORITIES statement is used, transaction codes with blocking calls (e.g. the KDCS call PGWT) can be assigned any TAC class as long as the TASKS-IN-PGWT operand of the MAX statement is generated with a value > 0. You must generate TAC PGWT=YES for these transaction codes.



If no TACCLASS statement or TAC-PRIORITIES statement is issued in the generation although the TACCLASS parameter was specified for at least one TAC statement, then the default values of the TACCLASS statement are applied. TAC priorities are not used in this case. See the TACCLASS description on [page 500](#) for more information.



You will find a detailed description of the TAC classes and priority control in [section “Job control - priorities and process limitations” on page 208](#).

```
TAC-PRIORITIES_ [ DIAL-PRIO={ABS | REL | EQ } ]
                [ ,ASYN-PRIO = { ABS | REL | EQ } ]
                [ ,FREE-DIAL-TASKS = number ]
```

DIAL-PRIO = Specifies according to which priority free processes will be distributed amongst the dialog TAC classes with waiting jobs. Waiting dialog jobs can only arise when more jobs are obtained from the job bourse at a specific time than there are processes available for the dialog TAC classes. The jobs are then written to the job queues of the transaction codes from which they will then be read out and processed according to their priority by the processes that become free.

ABS Absolute priority:
A free process is always assigned the TAC class with the highest priority (TAC class 1) as long as there are jobs waiting for this TAC class. TAC classes with lower priority are only serviced if there are no more jobs waiting in all TAC classes with higher priority.

REL Relative priority:
Free processes are assigned TAC classes with higher priority more often than TAC classes with lower priority as long as there are jobs waiting for the TAC classes with higher priority. If jobs are available for all dialog TAC classes, then a free process will be assigned TAC class 1 twice as often as TAC class 2, and TAC class 2 will be assigned processes twice as often as TAC class 3, etc.

EQ Equal priority:
As long as there are jobs available, all TAC classes are services equally often. This equal distribution can be interrupted if a TAC class does not have any jobs waiting for a while or when program unit runs with blocking calls (e.g. the KDCS call PGWT) often arise.

Default: EQ

ASYN-PRIO= Specifies according to which priority processes will be distributed amongst the asynchronous TAC classes with outstanding asynchronous jobs or interrupted asynchronous jobs.

If the maximum number of simultaneously open asynchronous services is reached (set in `MAX ASYNTASKS=(...,service_number)`), then no more asynchronous jobs are started. An interrupted open asynchronous service is selected according to its priority and resumed.

ABS Absolute priority:
 A free process is always assigned the TAC class with the highest priority, i.e. TAC class 9, as long as there are asynchronous jobs or interrupted asynchronous jobs waiting for this TAC class. Free processes only process the jobs of a TAC class with a lower priority when there are no more outstanding or interrupted asynchronous jobs in the message queues of all TAC classes with higher priority.

REL Relative priority:
 Free processes are assigned TAC classes with higher priority more often than TAC classes with lower priority as long as there are outstanding or interrupted jobs waiting for the TAC classes with higher priority. If jobs are available for all TAC classes, then a free process will be assigned TAC class 9 twice as often as TAC class 10, and TAC class 10 will be assigned processes twice as often as TAC class 11, etc.

EQ Equal priority:
 As long as there are jobs available, all TAC classes are services equally often. This equal distribution can be interrupted if a TAC class does not have any jobs waiting for a while or when program unit runs with blocking calls (e.g. the KDCS call PGWT) often arise.

Default: EQ

FREE-DIAL-TASKS=number

With FREE-DIAL-TASKS you limit the total number of processes that may process jobs to dialog TAC classes relative to the number of all processes of the application. In *number* you specify the minimum number of processes of the application that are to be reserved for processing jobs that do not belong to any dialog TAC class.



The maximum number of processes that may simultaneously process asynchronous jobs is not limited by FREE-DIAL-TASKS=. The MAX operand ASYNTASKS=*atask_number* is provided for this purpose.

Minimum value: 0 (no limit)

Maximum value: TASKS - 1 (TASKS from the MAX statement)

Default value: 1

Example

TASKS=7 and ASYNTASKS=2 was set in the MAX statement. FREE-DIAL-TASKS=3 is generated in the TAC-PRIORITIES statement. The application is operated with six processes. A maximum of three processes may process jobs in TAC classes 1 through 8 then. A maximum of two processes can process jobs in TAC classes 9 through 16. One process is reserved for dialog jobs that are not assigned a TAC class.

TCBENTRY - define a group of TCB entries (BS2000/OSD)

B The TBCENTRY control statement is permitted only for COB1 program units. COBOL
 B program units that are not ILCS-compatible must be generated with
 B PROGRAM ...,COMP=COB1 in openUTM. TCB entries offer benefits when used in
 B conjunction with COBOL-DML, and in the case of a GOTO in a PERFORM routine. Further
 B information can be found in the openUTM manual "Programming Applications with KDCS".

B TCB entries are used to create nested reentrant COBOL programs. They are required in
 B the following cases:

B ● In conjunction with COBOL-DML:
 B if the USE-DATABASE-EXCEPTION clause is used in a DECLARATIVES subsection and
 B the program run is terminated within these declaratives using PEND. In this case, you
 B must specify the TCB entry I\$ITCUPS; otherwise, the DECLARATIVES counter will not
 B be reset, resulting in a COBOL error action. I\$ITCUPS therefore resets the counter if a
 B PEND occurs within the declaratives.

B ● In the case of a GOTO in a PERFORM routine:
 B If a program unit is terminated in a PERFORM routine, the COBOL runtime system
 B notes the return address. If you branch to the PERFORM routine using GOTO in the
 B next program unit, the program unit behaves as if the PERFORM routine were still open
 B and branches to the return address. Markers are reset by specifying a TCB entry (with
 B any name).

B TCB entries must also be made known to the COB1 compiler using the COBRUN
 B parameter.

B The TCBENTRY statement can be issued several times.

B TCBENTRY_ tcbentry_groupname
 B ,ENTRY=(entry1,..., entry18)

B tcbentry_groupname
 B Freely selectable name up to eight characters in length, which is used to
 B address the group of TCB entries defined in this TCBENTRY statement,
 B and to link the group of TCB entries to a TAC statement.

B ENTRY= TCB entry name.

TLS - define a name for a TLS block

Each LTERM partner can be assigned a terminal-specific long-term storage area (TLS), which can contain several blocks. The TLS control statement allows you to define a name for a TLS block. The TLS block is then identified using the name of the LTERM partner (*ltermname*) and the block name defined here. openUTM provides each LTERM partner with a TLS block with this name. By issuing several TLS statements with different block names, you can define several blocks for each LTERM partner.

In the case of distributed processing, the TLS blocks defined in a TLS statement are also assigned to LPAP and OSI-LPAP partners.

You can issue up to 100 TLS statements.

TLS_ name

name Name of a TLS block up to eight characters in length

TPOOL - define an LTERM pool

The TPOOL control statement allows you to define the name and properties of an **LTERM pool**. LTERM pools allow you to connect numerous clients with the same technical properties (partner and processor type) to a UTM application via LTERM partners. Printers are not supported in this case. The TPOOL statement merely defines the type (PTYPE=) and processor name (PRONAM=) for the client. The LTERM partner assigned to the client is specified dynamically in the UTM system code during connection setup on the basis of the LTERM partner name and client name defined in the TPOOL statement. This assignment applies only for the duration of a session, i.e. it is not a static assignment as in the case of the statement pair LTERM / PTERM. The clients contained in an LTERM pool need not be configured explicitly in the application (by defining a PTERM). The number of clients that can be simultaneously signed on is equal to the number of LTERM partners generated in the LTERM pool.

For clients that connect via an LTERM pool (i.e. that are not explicitly generated), the establishment of the connection can only be initiated "from outside", i.e. from the client itself. It is therefore not possible to establish a connection via UTM administration commands. Also it is not possible to establish a connection via BCAM administration commands or using predefined BCAM connections under BS2000/OSD.

The TPOOL statement allows you to define LTERM pools with different levels of availability for connection setup:

- With PRONAM=*processormame* and PTYPE=*partnertyp*, the LTERM pool is generated such that only clients of the same type located on the specified system can establish connections with the UTM application via this LTERM pool.
- With PRONAM=**ANY*, all clients of a particular type can sign on to the UTM application irrespective of the system on which they are located.
- Under BS2000/OSD with PTYPE=**ANY*, you can define an LTERM pool without specifying the client type. This LTERM pool can then be used by clients of all types located on the specified system to establish connections with the UTM application.
- With PRONAM=**ANY* and PTYPE=**ANY*, all clients on all systems can sign on to the UTM application under BS2000/PSD (open LTERM pool).

It is possible to define several LTERM pools, i.e. issue several TPOOL statements in each KDCDEF run. However, please note the following:

- B ● *BS2000/OSD:*
 - B The combination PRONAM / PTYPE / BCAMAPPL must be unique for LTERM pools for which the NEABT user protocol is defined (PROTOCOL=STATION). The combination PRONAM/BCAMAPPL must also be unique for LTERM pools with PROTOCOL=NO.
 - B The client must support the user services protocol specified in the TPOOL statement. PROTOCOL=NO must be generated for clients with PTYPE=APPLI, PTYPE=SOCKET or PTYPE=UPIC-R. PROTOCOL=STATION must be specified for LTERM pools generated with PTYPE=*ANY.
 - B During connection setup, openUTM takes the type (PTYPE) of the client generated with PTYPE=*ANY from the user services protocol (connection letter). openUTM then checks whether or not this client type is supported. If not, the connection request is rejected.
- X/W ● *Unix systems and Windows systems:*
 - X/W The combination PRONAM/PTYPE/BCAMAPPL must be unique for LTERM pools.

The LTERM partners of an LTERM pool are generated with LTERM ..., RESTART=NO. During connection setup, therefore, all messages buffered in the message queue of the LTERM partners of the LTERM pool are deleted. An LTERM-specific service restart is not performed. In applications generated without user IDs, a service restart cannot be executed after a connection is cleared and then re-established for clients that are connected to the application via an LTERM pool.

You can specify access rights for an LTERM pool (KSET operand) that clients connected through the LTERM pool may exercise. In applications with user IDs you can limit the access rights specified with KSET for LTERM pools generated for connecting UPIC clients or TS applications using the USER-KSET operand. The access rights in KSET are then applicable to clients that explicitly specify a user ID when signing on. The limited access rights in USER-KSET take effect when the client does not specify a user ID when signing on, i.e. when the “connection user ID” is active.

B Using the **LOCALE** operand, you can define a client-specific language environment for each LTERM pool.

```
TPOOL_      [ ,BCAMAPPL=local_applname ]
            [ ,CONNECT-MODE={ SINGLE | MULTI }
            [ ,ENCRYPTION-LEVEL={ NONE | 1 | 2 | 3 | 4 | TRUSTED } ]
            [ ,IDLETIME=time ]
            [ ,KSET=keysetname1 ]
            [ ,LOCK=lockcode ]
            ,LTERM=ltermprefix
            [ ,MAP={ USER | SYSTEM | SYS11 | SYS21 | SYS31 | SYS41 } ]
            ,NUMBER=number1
            ,PRONAM={ processorname | C'processorname' | *ANY }
                    only mandatory in BS2000/OSD
            ,PTYPE={ partnertyp | *ANY2 }
            [ ,QLEV=queue_level_number ]
            [ ,STATUS=( { ON | OFF }[, number2 ] ) ]
            [ ,TERMN=termn_id ]
            [ ,USER-KSET=keysetname2 ]
            [ ,USP-HDR={ALL | MSG | NQ}]
```

B ***BS2000/OSD specific operands***

B [ANNOAMSG={ Y | N }]

B [,FORMAT= { + | * | # }formatname]

B [,KERBEROS-DIALOG={ YES | NO }]

B [,LOCALE=([lang_id][, [terr_id][, ccsname]])]

B [,NETPRIO={ MEDIUM | LOW }]

B [,PROTOCOL={ N | STATION }]

B ¹ The values SYS1, SYS2, SYS3 and SYS4 are only permitted under BS2000/OSD .

B ² *ANY is only permitted under BS2000/OSD.

B B B B	ANNOAMSG=(announce asynchronous message)	This applies only to LTERM pools used by terminals to sign on to the UTM application. It defines whether or not openUTM announces asynchronous messages before outputting them in the system line on the terminal.
B B	Y	Asynchronous messages are announced in advance. The user must then request the message using the KDCOUT command.
B		Default: Y
B B B B	N	Asynchronous messages are sent without prior announcement. If ANNOAMSG=N is generated, OMNIS V7.0 or later is required to establish a connection to terminals in this LTERM pool via a multiplex connection (see the description of the MUX statement on page 414).

BCAMAPPL=local_appliname

Name of the local UTM application. This name is then used to establish a connection between the client and the UTM application. *local_appliname* is defined either with MAX ...,APPLINAME= or in the BCAMAPPL statement (see [page 291](#)).

If you specify a value other than APPLI, SOCKET or UPIC-R for the PTYPE= operand, you can only specify the name defined with MAX ...,APPLINAME=*appliname* for *local_appliname*.

The BCAMAPPL name specified in the CLUSTER statement is not permitted here.

Default: *appliname*, specified under MAX ...,APPLINAME=.

B		<i>BS2000/OSD:</i>
B B B		The BCAMAPPL statement allows you to define whether or not NEA or ISO transport protocols or native TCP/IP (socket interface) are to be used when communicating with partners that sign on to this application.
B B		To establish a connection with the UTM application, the client must generally specify <i>local_appliname</i> as the partner name.
B B B B		One exception are LTERM pools that are generated with PTYPE=SOCKET. In this case, clients that connect via the LTERM pool must know the port number on which the UTM application "listens". This port number is specified in BCAMAPPL LISTENER-PORT= .

CONNECT-MODE=

This defines whether a client can use the same name for multiple sign-ons to the UTM application via this LTERM pool.

SINGLE Multiple sign-ons via the LTERM pool under the same name are not permitted.

Default: SINGLE

MULTI This is permitted only for LTERM pools that are used by UPIC partners or TS applications to connect.
A UPIC client program (PTYPE=UPIC-R or UPIC-L) or TS application (PTYPE=APPLI or SOCKET) can connect several times to the UTM application via the LTERM pool under the same name. A new name need not be created for each connection.

A UPIC client or TS application can connect a maximum of *number1* times to the LTERM pool (see NUMBER=*number1*, [page 522](#)).

In the case of CONNECT-MODE=MULTI, the UTM application does not identify the communication partner or the connection to the partner (as usual) using the name of the partner that the partner specified when the connection was established. The UTM application does not even know the partner under its application name. Instead, the partner is identified using the name of the pool LTERM partner (*ltermname*) through which it is connected. In order for openUTM to be able to uniquely identify the partner, the triplet consisting of the *ltermname* of the LTERM pool, the *processorname* and the *local_appliname* must not be explicitly generated in any PTERM, CON or OSI-CON statement. Additionally, the name that the partner specifies when establishing the connection may not match any LTERM name of the LTERM pool.

ENCRYPTION-LEVEL=

Only relevant for UPIC clients that support encryption and under BS2000/OSD for some terminal emulations that support encryption also.

In ENCRYPTION-LEVEL you set the minimum encryption level for the communication with the clients, that connect via an LTERM pool with the application.

You specify whether or not the UTM application should request encryption of the message on the connection via the LTERM pool to the client. You can also define the client as a "trusted" client. This means that every client that connects via this LTERM pool is considered to be a trusted client.(see also [section "Message encryption on connections to clients" on page 228](#) for more information on encryption).

You can specify the following:

NONE Encryption of the messages exchanged between the client and the UTM application is **not** requested by openUTM by default. Passwords are transmitted in encrypted form provided both partners support encryption with the longest available key, i.e. with AES if an RSA key with a length of ≥ 512 bits is available; otherwise with DES. The AES key is also encrypted with the longest available RSA key. Services for which encryption was generated for their service TACs (see ENCRYPTION-LEVEL in the TAC statement starting on [page 483](#)) can only be started by this client if the client explicitly selects an encryption level that corresponds to at least the required level when establishing the conversation or connection.

Default: NONE

1 | 2 | 3 | 4 Messages exchanged between the client and the UTM application are encrypted by openUTM by default. The value (1 to 4) specifies the encryption level. Only clients that support at least this encryption level can connect via this LTERM pool. If a client does not support the specified encryption level, openUTM rejects connection setup to the client.

Values 1 to 4 have the following meaning:

- 1 Passwords and input/output messages are encrypted using the DES algorithm. An RSA key with a key length of 200 bits is used to exchange the DES key.
- 2 Passwords and input/output messages are encrypted using the AES algorithm. An RSA key with a key length of 512 bits is used to exchange the AES key.
- 3 Passwords and input/output messages are encrypted using the AES algorithm. An RSA key with a key length of 1024 bits is used to exchange the AES key.
- 4 Passwords and input/output messages are encrypted using the AES algorithm. An RSA key with a key length of 2048 bits is used to exchange the AES key.

ENCRYPTION-LEVEL=1...4 makes only sense for UPIC partner, if the encryption functionality of openUTM is installed on your system. Otherwise clients cannot connection via the LTERM pool.

VTSU encryption is used for VTSU partners.

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 under BS2000/OSD. Levels 2, 3 and 4 are changed to 1 by KDCDEF without issue of a message.
- Encryption level 1, 2, 3 or 4 is changed to TRUSTED without issue of a message for local UPIC clients (PTYPE=UPIC-L) of an application under Unix systems or Windows systems.
- If 1 to 4 is specified for a partner of another type, the value is changed to NONE by openUTM without issue of a message.

B
B
B

X/W
X/W
X/W



If the application is generated with OPTION GEN-RSA-KEYS=NO, no RSA keys are created in the KDCDEF run. In order to use the encryption functions, you must create the required keys using administration facilities (KC_ENCRYPT or WinAdmin) or transfer them from an old KDCFILE using KDCUPD.

TRUSTED Messages between the client and the application are not encrypted. A "trusted" client can also start services whose service TACs request encryption (generated with TAC ENCRYPTION-LEVEL=1 or 2). This means that every client that connects via this LTERM pool is considered to be a trusted client.

TRUSTED should only be selected for an LTERM pool if all of the "physical" clients belonging to the pool are not accessible for everyone and communication is conducted through a secure connection.



Unix systems and Windows systems:

TRUSTED is the only value allowed for local UPIC clients (UPIC-L). Any other specification is changed to TRUSTED by openUTM without announcement.

X/W
X/W
X/W
X/W

FORMAT=

Start format for users on terminals that sign on to the application via this LTERM pool (see also the statement LTERM ...,FORMAT=, on [page 358](#)). Once the connection is established, the format specified in *formatname* is output on the terminal, provided a terminal-specific restart has not been performed.

B
B
B
B
B

Default: No start format

B

IDLETIME=*time*

The maximum time in seconds that openUTM may wait for input from the client outside of a transaction, i.e. after the end of a transaction or after signing on. If this time is exceeded, then openUTM clears down the connection to the client. If the client is a terminal, then message K021 is output before the connection is cleared.

The value specified for *time* may not be smaller than the value of the timer TERMWAIT=*time* (wait time between dialog output and response within a multi-step transaction; see [page 400](#)) and PGWTTIME=*time* (time that a program unit waits for an incoming message after a blocking call; see [page 391](#)) that you specified in the MAX statement.

This function serves to improve data security:

If a user forgets to sign off from the terminal when taking a break or when finishing his or her work on the terminal, then the connection to the terminal or client is automatically cleared down after the wait time has run out. This reduces the chance of someone gaining unauthorized access to the system.

Default: 0 (= no wait time limit)

MAX TERMWAIT=(...,*time2*) is used for terminals (when it is set).

Maximum value: 32767

Minimum value: 60

If you specify a value that is smaller than the minimum value, then KDCDEF replaces the value with the minimum value.

B KERBEROS-DIALOG =

B Y A Kerberos dialog is performed when a connection is established for
B terminals that support Kerberos and that connect to the application directly
B via this terminal pool (not via OMNIS).

B openUTM stores the Kerberos information in the length resulting from the
B maximum lengths generated for MAX PRINCIPAL-LTH and MAX
B CARDLTH. If the Kerberos information is longer, it is truncated to this length
B and stored.

B If a length greater than zero is generated neither for MAX PRINCIPAL-LTH
B nor for MAX CARDLTH, a warning message is issued.

B The KDCS call INFO (KCOM=CD) allows a program unit run to read this
B information.

B Exception: A user has subsequently signed on to this client with an ID card.
B In this event, the Kerberos information is overwritten by the card ID
B information.

B B	N	<p>No Kerberos dialog is performed, Default.</p> <p>KSET=keysetname1 Name of a key set assigned to this LTERM pool. The key set must be defined with a KSET statement.</p> <p>This defines the access permissions for the LTERM partners of this LTERM pool with respect to using the services of the application and remote services (LTACs) generated in this application.</p> <p>An LTERM partner of this LTERM pool can only be used to start services of the application that are protected with a lock code or an access list and only address remote services that are protected with a lock code or an access list if the following applies: The key set assigned to the LTERM partner and the KSET of the UTM user ID under which sign-on using this LTERM partner was performed must contain the key code or access code that matches the lock code or access list.</p> <p>With PTYPE=APPLI, SOCKET, UPIC-R, UPIC-L the following additionally applies with respect to the key set of the user ID:</p> <ul style="list-style-type: none"> – If the client does not pass a real user ID to openUTM for the session/conversation, then its access rights are the result of the set of key codes that are contained in the key set generated with KSET and the key set generated with USER-KSET. The key set <i>keysetname1</i> should therefore contain all key codes that are also contained in the key set generated with USER-KSET. – If the client passes a user ID, then its access rights are the result of the set of key codes that are contained in the key set of the user ID and the key set generated with KSET.
B B B	LOCALE=(lang_id,terr_id,ccsname)	<p>Language environment of clients that sign on to the UTM application via the LTERM pool.</p>
B B B B B	lang_id	<p>Freely selectable language identifier for the clients of the LTERM pool, up to two characters in length.</p> <p>The language identifier may be queried by the program units of the application, so that messages can be sent to the terminals in the client's language.</p>
B B	terr_id	<p>Freely selectable territorial identifier for the clients of the LTERM pool, up to two characters in length.</p>
B B B		<p>The territorial identifier may be queried by the program units of the application, so that any special territorial features of the client's language can be taken into consideration in messages.</p>

B	ccsname (coded character set name)
B	Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). The character set must be compatible with an ISO character set supported by all terminals in the LTERM pool.
B	During generation, KDCDEF cannot check the validity of the CCS name under the BS2000 system or the compatibility condition.
B	The character set with the specified name is used for:
B	– outputting dialog messages on 8-bit terminals if the application is generated without user IDs, or if a user has not yet signed on to the LTERM partner of the LTERM pool and another CCS name has not been explicitly selected using an edit profile or a format.
B	– outputting asynchronous messages on 8-bit terminals if another CCS name is not explicitly selected using an edit profile or a format.
B	Default: If TPOOL ...,LOCALE is not specified, then the locale of the application defined in the MAX statement is used.

LOCK=lockcode

Access protection to the LTERM pool. Lock code assigned to the LTERM partners of the LTERM pool. *lockcode* is a numeric value between 1 and the maximum value permitted in the application (MAX ...,KEYVALUE=). You can only sign on to the application on an LTERM partner of this LTERM pool under a UTM user ID (USER) for which a key set was generated with a key code that matches the lock code of the LTERM pool.

Default: 0 (the LTERM pool is not secured with a lock code)

Maximum value: Value of KEYVALUE defined in the MAX statement

LTERM=ltermprefix

Prefix for the names of LTERM partners of the LTERM pool. LTERM names are eight characters in length, and consist of the prefix specified here followed by a serial number between 1 and the value defined for NUMBER=*number1*.

The maximum length of *ltermprefix* depends on the number of decimal places in *number1*. The number of characters in *ltermprefix* plus the number of decimal places in *number1* must be less than 8.

When specifying *ltermprefix* and *number1*, please note that LTERM partner names must be unique within the application. This applies for names generated with TPOOL ...,LTERM= (in all TPOOL statements) and for names defined in LTERM statements.

Example

With *number1*=1000 and LTERM=LTRM, the LTERM partners defined for the LTERM pool are assigned the names LTRM0001,LTRM0002,...,LTRM1000.

These names must not be specified in any LTERM statement.



The specified names must not be assigned to any other object in name class 1. See also [section "Uniqueness of names and addresses" on page 265](#).

MAP=

In MAP you specify whether or not openUTM is to convert the code of user messages exchanged with the communication partner (EBCDIC <->ASCII). User messages are passed in the message area on the KDCS interface in the message handling calls (MPUT/MGET/FPUT/DPUT/FGET).

MAP= controls the conversion when exchanging unformatted messages with other applications. openUTM does not generally execute any message handling for formatted messages.

B

BS2000/OSD:

B

MAP≠USER is only permitted for LTERM pools with PTYPE=SOCKET.

X/W

Unix systems and Windows systems:

X/W

MAP≠USER is permitted for all TS applications (PTYPE=APPLI or SOCKET).

X/W

USER

openUTM does not convert the data of the message area, i.e. the messages are transferred to the partner application unchanged and the messages received from the partner are transferred unchanged to the program unit. Note that the user message contains the transaction code. It must be encoded in the form that the receiving system expects, i.e. under BS2000/OSD in EBCDIC and in ASCII under Unix systems and Windows systems.

Default: USER

B

SYSTEM / SYS / SYS1 / SYS2 / SYS3 / SYS4 (BS2000/OSD)

B

These may only be specified when the messages received by the TS application are not encoded in EBCDIC, or when the TS application expects messages encoded in ASCII from the UTM application.

B

B

B

If you specify one of the values above, then openUTM converts the data in the message from EBCDIC to ASCII before the message is sent and from ASCII to EBCDIC after receiving a message. openUTM assumes that the messages only contain printable characters when converting back and forth.

B

B

B

B

B B B B		You specify the conversion table to be used by openUTM for the code conversion with SYSTEM, SYS, SYS1, SYS2, SYS3, SYS4. The conversion tables must be defined in the module KDCEA (see page 613).
B B B B		SYSTEM, SYS and SYS1 are synonyms. If you specify one of these values, then openUTM uses the standard code table for the conversion that converts EBCDIC to 7-bit ASCII. The standard code table is already defined in KDCEA and can be used without any additional preparation.
B B B B		If openUTM is to execute a different conversion from EBCDIC to ASCII, then you must define the corresponding conversion table yourself in KDCEA. You can define up to a maximum of three conversion tables in KDCEA (Table 2 through Table 4).
B B B		If you specify SYS2, then openUTM converts the user messages using table 2. openUTM uses table 3 when SYS3 is specified and table 4 when SYS4 is specified.
X/W X/W X/W X/W X/W X/W	SYSTEM	(Unix systems and Windows systems) openUTM converts the data in the KDCS message area from ASCII to EBCDIC before sending messages, or from EBCDIC to ASCII after receiving messages. Messages must contain printable characters only and must be created in line mode (KCMF = blank). The parameter is only allowed for PTYPE=APPLI or SOCKET.
B B	NETPRIO=	Transport priority to be used on the transport connections assigned to this LTERM pool.
B B		NETPRIO is not relevant when the connection from the partner application is established via the socket interface (transport protocol SOCKET).
B		Default: MEDIUM
	NUMBER=number1	Maximum number of LTERM partners in this LTERM pool. Up to <i>number1</i> clients can then sign on to the application via the LTERM pool. The maximum value permitted for <i>number1</i> depends on the number of names generated in the UTM application (see section "Number of names" on page 262). Minimum value: 1

	PRONAM=	System on which the clients must be located in order to sign on to the application via this LTERM pool.
B		<i>BS2000/OSD:</i>
B		PRONAM= is a mandatory operand.
B		If PROTOCOL=STATION is set, the combination of PRONAM/PTYPE/BCAMAPPL must be unique.
B		If PROTOCOL=NO is set, the combination of PRONAM/BCAMAPPL must be unique.
X/W		<i>Unix systems and Windows systems:</i>
X/W		PRONAM= may only be specified for LTERM pools of type PTYPE=APPLI, SOCKET or UPIC-R.
X/W		The combination of PRONAM/PTYPE/BCAMAPPL must be unique.
X/W		Default value under Unix systems and Windows systems: 8 blanks
	{ processorname C'processorname' }	Host name, up to eight characters in length. Only clients connected to this system can sign on to the application via this LTERM pool. If <i>processorname</i> contains special characters it must be entered in the form of a character string using C'...' .
		If an LTERM pool is generated using PTYPE= SOCKET, then you must specify the symbolic address (official host name) under which the IP address of the host partner is entered in the name service of the local system (e.g. the hosts file under Unix systems and Windows systems or the RDF file under BS2000/OSD) for <i>processorname</i> . You must not specify an alias of the host.
X/W		<i>Unix systems and Windows systems:</i>
X/W		There are two options for specifying the <i>processorname</i> :
X/W		– You enter the real host name under which the IP address of the partner computer is entered in the name service of the local system (e.g. the hosts file). You must not specify an alias of the computer.
X/W		– You enter the UTM host name of the partner computer.
X/W		This is only possible when you have set the UTM_NET_HOSTNAME environment variable and specified the UTM host name in the conversion file (see the section “Using mapped host names (Unix systems and Window systems)” on page 122).
X/W		
X/W		
X/W		
X/W		

	*ANY	Any client that fulfills the following conditions can sign on to the application via the LTERM pool: <ul style="list-style-type: none"> – The client must not be explicitly generated in a PTERM statement. – The client type must match the entry in PTYPE. – Another LTERM pool must not be generated for the system on which the client is located or for the same client type. This prevents open LTERM pools from being used as an “overflow” for other LTERM pools.
B B B	PROTOCOL=	This specifies whether or not the user services protocol (NEABT) is to be used between the UTM application and the clients accessed via this LTERM pool.
B B B B B B B	N	openUTM does not use a user services protocol. If PROTOCOL=N is generated, it is not possible to establish connections to terminals in this LTERM pool via a multiplex connection (see the description of the MUX statement on page 414). PROTOCOL=N must be generated for UPIC client programs (PTYPE=UPIC-R) and for TS applications (PTYPE=APPLI or SOCKET). In this case, openUTM ignores the entry PROTOCOL=STATION without outputting a UTM message.
B		If you specify PTYPE=*ANY, openUTM ignores the entry PROTOCOL=NO.
B B	STATION	The user services protocol (NEABT) is used between the UTM application and the clients accessed via this LTERM pool.
B B B B		With PTYPE=*ANY, you must specify PROTOCOL=STATION. In this case, openUTM requires the user services protocol (NEABT) to determine the partner type if this is not explicitly specified during generation (PTYPE=*ANY).
B B B		Default: N if PTYPE=APPLI, SOCKET or UPIC-R STATION if PTYPE≠APPLI, SOCKET or UPIC-R.
	PTYPE=	Type of client that can sign on to the application via this LTERM pool. If you have specified an application name in BCAMAPPL= that is generated for communication via the socket interface (BCAMAPPL statement with T-PROT=SOCKET), then you must set PTYPE=SOCKET. This is a mandatory operand.
	partnertyp	Type of client. A list of partner types supported can be found in the description of the PTERM control statement on page 450 (BS2000/OSD) and on page 453 (Unix systems and Windows systems). Please note that printers cannot be connected via LTERM pools.

B B B B	*ANY	<p>only permitted under BS2000/OSD.</p> <p>PTYPE=*ANY describes an open LTERM pool. All clients that support the user services protocol (PROTOCOL=STATION) and that are located on the processor defined with PRONAM= may connect to this LTERM pool.</p> <p>In this case, openUTM takes the partner type from the user services protocol during connection setup. Only then can it be determined whether or not the partner type is supported.</p> <p>The advantage of PTYPE=*ANY is that it allows you to include clients in the configuration without having to know how they are generated in PDN. The configuration is also easier to maintain if, following regeneration in PDN, these clients can still sign on to the application without having to modify the KDCDEF generation.</p>
	QLEV=queue_level_number	<p>(queue level)</p> <p>Maximum number of asynchronous messages that can be accommodated in the message queue of the LTERM partner. If this threshold value is exceeded, openUTM rejects all further FPUT calls for this LTERM partner with UTM message 40Z.</p> <p>Default: 32767 Minimum value: 0 Maximum value: 32767</p> <p>If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.</p>
	STATUS=	<p>NUMBER=<i>number1</i> defines the number of LTERM partners in the LTERM pool. STATUS=<i>number2</i> defines the number of clients that are unlocked (ON) and locked (OFF) for the LTERM pool when the application is started. The status can be modified by administration during operation.</p> <p>Default: STATUS=(OFF , 0), i.e. all clients of the LTERM pools are unlocked.</p>
	ON	<i>number2</i> clients are unlocked.
	OFF	<i>number2</i> clients are locked.
	number2	Number of clients (and thus the number of LTERM partners of the LTERM pool) which are locked or unlocked.

TERMN=termn_id

Identifier up to two characters in length, which indicates the type of client. openUTM provides this identifier to the application program in the KCTERMN field of the KB header.

termn_id is not queried by openUTM, but can be used by the user for analysis purposes.

Default values:

If this operand is not specified, openUTM sets the KCTERMN field to the default ID of the partner type specified in the PTYPE operand. However, the user can select other values if desired.

B
B
B
B
B
B
B
B

– *BS2000/OSD:*

The default values are listed in the partner type table for the PTYPE= operand of the PTERM statement [on page 451](#).

If TERMN is not explicitly specified for clients generated with PTYPE=*ANY, openUTM does not enter the terminal mnemonic in KCTERMN until the connection is established. This is the default terminal mnemonic of the type specified in the user services protocol of the connection request.

X/W
X/W

– *Unix systems and Windows systems:*

The default values are listed in the table below.

X/W
X/W
X/W
X/W
X/W
X/W

PTYPE	TERMN
TTY	F1
APPLI	A1
UPIC-L	A2
UPIC-R	A5
SOCKET	A7

USER-KSET=keysetname2

This is only allowed if the application is generated with user IDs and PTYPE=APPLI, SOCKET, UPIC-R or UPIC-L is specified. You may only set USER-KSET= in conjunction with KSET= .

You must specify the name of a key set for *keysetname2*. The key set must be defined with a KSET statement.

You specify the minimum access rights that a client connected via this LTERM pool can exercise with USER-KSET= .

keysetname2 takes effect when the client is signed on under the connection user ID. Its access rights are the result of the set of key codes that are contained in the key set generated with KSET= and in the key set generated with USER-KSET= (intersection). For this reason, all key codes contained in USER-KSET=*keysetname2* should also be contained in KSET=*keysetname1*.

Default: No key set

The access rights specified in KSET are always valid.

USP-HDR=

Specifies the output messages for which openUTM is to create a UTM socket protocol header for the connections generated with this statement.

A value that is not equal to NO may only be specified with LTERM pools for which communication is configured via socket connections (PTYPE=SOCKET).

A description of the USP header can be found in the openUTM manual "Programming Applications with KDCS".

ALL

For all output messages (dialog, asynchronous, K messages) openUTM creates a UTM socket protocol header and adds this to the front of the message.

MSG

openUTM only creates a UTM socket protocol header and adds this to the front of the message for K messages only.

NO

No UTM socket protocol headers are created.

Default: NO

TRANSFER-SYNTAX - define the transfer syntax

You only need the TRANSFER-SYNTAX control statement when you want to define your own application context for communication based on the OSI TP protocol (see the APPLICATION-CONTEXT statement on [page 285](#)).

It allows you to define a local name for a transfer syntax, and to assign an object identifier. The transfer syntax determines the rules governing the encoding and decoding of the abstract syntax defined in the object identifier.

The transfer syntax defined here must be supported by the OSS version used. OSS only supports the BER transfer syntax at the present time.

```
TRANSFER-SYNTAX_ transfer_syntax_name
                    ,OBJECT-IDENTIFIER=object_identifier
```

transfer_syntax_name

Local name for a transfer syntax up to eight characters in length. This name must be unique within the UTM application.

The *transfer_syntax_name* BER (Basic Encoding Rules) is reserved.

OBJECT-IDENTIFIER=object_identifier

Object identifier of the transfer syntax specified as follows:

object_identifier=(number1,number2, ... ,number10)

number is a positive integer in the range 0 to 67108863. For *object_identifier*, you can specify two to ten integers enclosed in parentheses, each of which is separated by a comma. The number of integers entered and their positions are relevant.

Instead of the integer itself, you can also specify the symbolic name assigned to this integer. The table on [page 97](#) shows the permitted values for *number* at the various positions.

object_identifier must be unique within the UTM application, i.e. another transfer syntax must not be generated with the same object identifier.

openUTM generates the BER transfer syntax by default:

```
TRANSFER-SYNTAX_BER,
                    OBJECT-IDENTIFIER=(2, 1, 1)
                    -
                    -
```

Symbolic description of the object identifier:

(joint-iso-ccitt, ansi, basic-encoding)

ULS - define a name for a ULS block

Each UTM user ID can be assigned a user-specific long-term storage area (ULS), which can contain several blocks each of which is addressed by means of a name.

The ULS control statement allows you to define a name for a ULS block. openUTM then provides each UTM user ID with a ULS block with this name. By issuing several ULS statements with different block names, you can define several blocks.

In the case of distributed processing based on LU6.1, the ULS blocks defined in a ULS statement are also assigned to sessions (LSES).

This statement is required only if the application is generated with user IDs.

You can issue up to 100 ULS statements.

Note on UTM cluster applications

If you modify, remove or add ULS statements then you must regenerate both the initial KDCFILE and the UTM cluster files by specifying GEN=(CLUSTER,KDCFILE) in the OPTION statement.

ULS_	blockname
------	-----------

blockname	Name of a ULS block up to eight characters in length, which can be used to address the block from a program unit.
-----------	---

USER - define a user ID

The USER control statement allows you to define user IDs for the UTM application. These are then used by users and client programs to sign on to the application. The following can be defined for user IDs:

- the authentication procedure (password, magnetic strip card under BS2000/OSD and Unix)
- complexity level and period of validity of password
- access rights (lock/key code or access list concept)
- administration authorization
- the user status
- properties of the USER queue that belongs to the user ID
- B – the start format
- B – UTM SAT administration authorization (BS2000/OSD)
- B – the user-specific language environment (BS2000/OSD).
- B – Method and type of authentication (BS2000/OSD: password, magnetic strip card, Kerberos principal)

At least one user ID must be assigned administration authorization in order to manage the application. Administration authorization can be granted to several user IDs, thereby enabling several users to simultaneously call administration functions under the respective user ID. Under BS2000/OSD the same is true for the UTM SAT administration authorization and calling SAT preselection functions.

An application can also be generated without user IDs. In this case, users are not required to identify themselves, and openUTM uses the name of the respective client internally as the user ID. All users can thus issue administration commands and under BS2000/OSD UTM SAT administration commands. If you work without user IDs, openUTM will not be able to use some data protection functions.

```

USER_      username

           [ ,KSET=keysetname ]

           [ ,PASS={ ( password,DARK) |
                     ( *RANDOM,DARK ) |
                     password      |
                     *RANDOM } ]

           [ ,KSET=keysetname ]

           [ ,PERMIT={ NONE |ADMIN | SATADM1 | ( ADMIN,SATADM)1 }

           [ ,PROTECT-PW=( [ length ]
                           ,[ { NONE | MIN | MED | MAX } ]
                           ,[ maxtime ]
                           ,[ mintime ] ) ]2

           [ ,QLEV=queue_level_number ]

           [ ,QMODE = { STD | WRAP-AROUND } ]

           [ ,Q-READ-ACL=read-keysetname ]

           [ ,Q-WRITE-ACL=write-keysetname ]

           [ ,RESTART={ YES | NO } ]

           [ ,STATUS={ ON | OFF } ]

```

B

BS2000/OSD specific operands

B

```
[ ,CARD=( position,characterstring ) ]
```

B

B

```
[ ,CERTIFICATE={ *NONE |
                 number [,CERTIFICATE-AUTHORITY = number1 ] } ]
```

B

```
[ ,FORMAT= { + | * | # }formatname ]
```

B

```
[ ,LOCALE=( [ lang_id ][,[ terr_id ][ ,ccsname ] ] ) ]
```

B

```
[ ,PRINCIPAL=characterstring ]
```

B

```
[ ,SATSEL={ NONE | BOTH | SUCC | FAIL } ]
```

B

¹ only permitted under BS2000/OSD

² Commas at the end can be omitted, i.e. you can specify (8,NONE) instead of (8,NONE,).

username UTM user ID specified by the user when signing on to the application, or by a client when opening a conversation with the application. *username* can be up to eight characters in length.
(See also the operand USER=*username* in the LTERM statement starting on [page 355](#)).



The specified name must be unique and must not be assigned to any other object in name class 2. See also [section “Uniqueness of names and addresses” on page 265](#).

If *username* is identical to the name of an LTERM that is assigned to a PTERM with PTYPE=APPLI, SOCKET or UPIC-R, you must bear in mind the notes detailed under LTERM.

B
B
B
B
B

CARD=

(only allowed under BS2000/OSD)
specifies whether a magnetic strip card is to be checked when signing on to the application under this user ID, and defines the ID card information to be verified. Enter a subfield of the information stored on the magnetic strip card, which is to be checked by openUTM.

B
B
B

The following must apply for this parameter:
pos + length(string) - 1 ≤ MAX CARDLTH
Otherwise the parameter is ignored.

B
B

You cannot specify CERTIFICATE= or PRINCIPAL= if you have specified CARD=.

B
B

position

Start position of the ID card information to be checked:
position = 1 corresponds to the first byte, etc.

B
B
B

characterstring

When signing on to the application, openUTM checks whether the ID card information starting at the defined position begins with this character string.

B

characterstring can be specified in the following format:

B
B
B

- as a hexadecimal character string; hexadecimal characters always occur in pairs, e.g. X'DDEF'
- as an alphanumeric character string, e.g. FRIDOLIN or C'@FRIEDEL'.

B

Special characters must be entered in the format C'...' or X'...'.

B
B

Default: No ID card check performed when signing on to the application
Maximum length: 100 bytes (see also MAX ...,CARDLTH=)

B B B B B	CERTIFICATE= (only permitted in BS2000/OSD)	specifies whether a user is to sign on to a UTM application using a chip card which contains the generated certificate. When signing on openUTM checks to see whether the certificate it has received corresponds to the generated certificate. For more information see also openUTM manual "Concepts und Functions".
B B		You cannot specify CARD=, PASS= or PRINCIPAL= if you have specified CERTIFICATE=.
B	*NONE	The user is not required to identify themselves using a certificate.
B		Default: *NONE
B B	number	The user is required to identify themselves using a certificate. Here <i>number</i> is the number of the certificate that the user must provide.
B		Minimum: 0
B		Maximum: 2147483674
B	CERTIFICATE-AUTHORITY=number1	
B B B		The number of the certificate authority that issued the certificate. If <i>number1</i> is omitted or <i>number1</i> =0 specified, then the number of the certificate authority is not checked when the user signs on.
B		Default: 0 (= no check)
B		Minimum: 0 or 1 (for a real number)
B		Maximum: 2147483674
B B B B B B B	FORMAT=	Format identifier for a user-specific start format. This start format is automatically output after each successful attempt to sign on to the application, provided an open service does not exist for this user. However, if an open service exists for the user (USER) following a successful sign-on check, the start format is not displayed, rather the last dialog screen appears (service restart). If you use your own sign-on procedure, the name of the user-specific start format may be queried in the second part of the sign-on procedure using the SIGN ST call.
B		The format identifier composes as follows:
B B		+, * or # followed by an alphanumeric name (<i>formatname</i>) up to seven characters in length.
B		#formats can only be used in the context of a sign-on procedure.

B		The terms have the following meanings:
B	+	When the next MGET call of the program unit is issued, each entry in a format field is preceded by 2 bytes for the attribute field in the KDCS message area, i.e. the field properties can be modified by the program unit. The format identifier at the KDCS interface is thus <i>+formatname</i> .
B		
B		
B	*	When the next MGET call of the program unit is issued, the entry in a format field is not preceded by any bytes for an attribute field in the KDCS message area, i.e. the field properties cannot be modified by the program unit. The format identifier at the KDCS interface is thus <i>*formatname</i> .
B		
B	#	This identifies a format with extended user attributes. The field properties and global format properties can be modified by the program unit. The format identifier at the KDCS interface is thus <i>#formatname</i> .
B		
B		Default: No start format

KSET=keysetname

Name of the key set assigned to the user ID. The key set is defined with the KSET statement. A maximum of one key set can be assigned per USER.

The key set defines the access permissions for this user ID with respect to using the services of the application and remote services (LTACs) generated in this application.

This user ID can only be used to start services of the application that are protected with a lock code or an access list and only address remote services that are protected with a lock code or an access list if the following applies: The key set assigned to the user ID and the key set of the LTERM partner under which sign-on using this user ID was performed must contain the key code or access code that matches the lock code or access list.

The lock/key code concept and the access list concept are both described in detail in the openUTM manual “Concepts und Functions”. An introduction to data access control can be found as of [page 219](#).

Services whose service TACs are not secured with codes can be called by the user or the client program without restriction. Further information on the lock/key code concept can be found in the openUTM manual “Concepts und Functions”.

Default: No key set, i.e. the user can only access clients and LTERM partners that have not been secured with lock codes.

B B	LOCALE=(lang_id,terr_id,ccsname)	Language environment of the user
B B	lang_id	Freely selectable language identifier for the UTM user ID, up to two characters in length.
B B B		The language identifier may be queried by the program units of the application, so that messages can be sent by the program units to the user ID in the user's language.
B B	terr_id	Freely selectable territorial identifier for the UTM user ID, up to two characters in length.
B B B		The territorial identifier may be queried by the program units of the application, so that any special territorial features of the user's language can be taken into consideration in messages to the user.
B B B B B B	ccsname	<p>(coded character set name) Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the "XHCS User Guide"). During generation, openUTM cannot check the validity of the CCS name under the BS2000 system.</p>
B B B B B B		<p>The character set with the specified CCS name is used for outputting dialog messages, provided the user is signed on to an 8-bit terminal and another CCS name is not explicitly selected using an edit profile or a format. The character set must be compatible with an extended ISO character set supported by the terminal. During generation, openUTM cannot check this compatibility condition, i.e. incorrect entries cannot be intercepted by KDCDEF.</p>
B B B		<p>Default: Locale of the application defined in the MAX statement is used if USER ...,LOCALE is not specified.</p>

PASS= Password up to eight characters in length which must be specified by the user during the sign-on check. This password must comply with the level of complexity defined in the PROTECT-PW= operand.

B
B

PASS= may not be specified together with CERTIFICATE= or PRINCIPAL=.

If you enter *RANDOM here, a secret random password is generated for the user ID. A valid password must then be transferred to the user ID using the KDCUPD tool or by means of administration. Passwords created in this way are not subject to the conditions set in PROTECT-PW=.



Make sure that at least one user ID is configured with administration authorization by the startup time at the latest. This user ID must not be assigned a password created using *RANDOM, as the application cannot be administered otherwise.

The parameters have the following meanings:

B

BS2000/OSD:

B
B
B
B
B

password
*RANDOM

Standard sign-on dialog:

The user must enter 'KDCSIGN *username,password*' to sign on to the application.

B
B
B

Sign-on procedure:

The user ID and password must be transferred to openUTM using the SIGN ON call.

B
B
B
B
B
B

(password, DARK)
(*RANDOM,DARK)

Standard sign-on dialog:

To sign on to the application the user first enters 'KDCSIGN *username*'. openUTM then prompts the user to enter the password in a blanked-out field on the screen.

B
B
B

Sign-on procedure:

In an intermediate dialog, openUTM prompts the user to enter the password in a blanked-out field.

B
B
B
B

password can be entered in the following format:

- hexadecimal, e.g. X'DDFF'
- as a constant, e.g. C'@LKE'
- as a printable alphanumeric character string, e.g. NORBERT.

B

Default: 8 blanks (i.e. no password)

X/W		<i>Unix systems and Windows systems:</i>
X/W		password
X/W		*RANDOM
X/W		(password,DARK)
X/W		(*RANDOM,DARK)
X/W		Standard sign-on dialog:
X/W		To sign on to the application, the user first enters <i>username</i> . openUTM then prompts the user to enter the password.
X/W		
X/W		Sign-on procedure:
X/W		In an intermediate dialog, openUTM prompts the user to enter the password as in the standard sign-on procedure.
X/W		
X/W		The password can be entered in the form of a printable alphanumeric character string, e.g. UTM4EVER or C'UTM_4EVER'.
X/W		
X/W		Default: 8 blanks (i.e. no password)
	PERMIT=	Administration authorization level of the user within the local application
	ADMIN	The user can execute administration functions under this user ID.
	NONE	The user must not execute any administration functions.
		Default: NONE
B		Under BS2000/OSD the user also must not execute any SAT preselection functions.
B		
B	SATADM	The user can execute SAT preselection functions (UTM SAT administration).
B		
B	(ADMIN,SATADM)	The user can execute administration and SAT preselection functions.
B		
B	PRINCIPAL=characterstring	(only permitted in BS2000/OSD)
B		Authentication of the user is to be performed using Kerberos. It is only possible to authenticate users using Kerberos if the user signs in directly (not via OMNIS) at a terminal that supports Kerberos.
B		
B		openUTM stores the Kerberos information in the maximum of the lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated and stored in this length.
B		
B		The KDCS call INFO (KCOM=CD) enables a program unit to read this information as long as the user is signed in on this client.
B		
B		Specifying PRINCIPAL excludes the possibility of specifying the parameters CERTIFICATE, CARD, and PASS.

B B	<i>characterstring</i> must be specified as follows as an alphanumeric string enclosed in single quotes:
B	C'windowsaccount@NT-DNS-REALM-NAME'
B B	windowsaccount Domain account of the user
B B B B	NT-DNS-REALM-NAME DNS name of the Active Directory domain. This name is a fixed value for every Active Directory domain and was assigned when the Kerberos key was set up.
B B B	The length of the character string passed must not be greater than the value specified for MAX PRINCIPAL-LTH. Otherwise the parameter is ignored.
B B B B	openUTM stores the Kerberos information in the length resulting from the maximum length generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated to this length and stored.
B B	The KDCS call INFO (KCOM=CD) allows a program unit run to read this information as long as the user is signed in on this client.
B B B	Default: No Kerberos authentication Maximum length: The value generated with MAX ...,PRINCIPAL-LTH. See page 392

PROTECT-PW=

Specifies the minimum length, level of complexity, and minimum and maximum validity period of the user password. The values defined for PROTECT-PW must be taken into consideration when specifying the password in the PASS= operand. They are checked by openUTM when the password is changed by the administrator (KDCUSER administration command, see the openUTM manual “Administering Applications”) or by a program unit (SIGN CP call).

length

Minimum number of characters that must be contained in the password. The administrator can only delete the user password if the value 0 is specified for *length*.

Default: 0

Minimum value:

0 for NONE or if a level of complexity is not defined

1 for MIN

2 for MED

3 for MAX

Maximum value: 8

NONE/MIN/MED/MAX

Level of complexity of the password

NONE The password can be any character string.

Default: NONE

MIN In the password, up to two consecutive characters may be identical. The minimum length of the password is one character.

MED In the password, up to two consecutive characters may be identical. The password must contain at least one letter and one number. The minimum length of the password is two characters.

MAX In the password, up to two consecutive characters may be identical. The password must contain at least one letter, one number, and one special character. The minimum length of the password is three characters. Special characters are all characters other than a-z, A-Z, 0-9, and blanks.

maxtime Maximum validity period:

maxtime specifies the maximum number of days for which the password is valid.

If a validity period is specified, then the validity of the password expires at the end of the last day of the specified validity period. For instance, if the validity period is one day, the password ceases to be valid at 24:00 hours on the following day.

If the application is generated with SIGNON GRACE=YES, when the application is regenerated the password is set to “expired”, the user must then assign a new password the first time they sign on.

If the password expires, then the next action taken depends on how the UTM application is generated:

- With grace sign-ons (SIGNON GRACE= YES)
The user can and must change the password the next time they sign on, as long as the sign-on service of the application offers them this opportunity. If this is not the case, the password must be modified by administration otherwise the user will no longer be able to sign on under this user ID. This may occur, for example, with users that sign on via TS applications and UPIC clients without sign on services or via an OSI-TP partner.
- Without grace sign-ons (SIGNON GRACE= NO)
openUTM rejects a sign-on attempt with message K120. The administrator must then change the password.

With *maxtime* = 0 the validity period of the password is not restricted.

Default: 0 (validity period not restricted)

Maximum value: 180

Minimum value: 0

mintime

Minimum validity period:

You specify the minimum validity period of the password in days in *mintime*. Once the user has changed the password, the user may only change the password again after the minimum validity period has expired.

By specifying *mintime* > 0 you can prevent a user whose password has expired from changing his or her password twice in a row to set the password back to the original (= expired) password.

If a minimum validity period of one day is specified, then the password may be changed no earlier than at 12.00 midnight of the following day (local time of the generation).

The user can always change the password after the administrator has changed the password and after a new generation, regardless of whether the minimum validity period has expired or not.

mintime must not be larger than *maxtime* (maximum validity period).

If *mintime*=0 is specified, then the minimum validity period of the password is not restricted.

Default: 0 (no limit)

Minimum value: 0

Maximum value: 180

QLEV=queue_level_number

(**queue level**)

Specifies the maximum number of asynchronous messages that may be buffered in the message queue of the user (= USER queue). QLEV can be used to make sure that the page pool is not overloaded with messages for this USER.

openUTM only takes asynchronous jobs into account at the end of the transaction. It is thus possible that the maximum number of messages for a message queue as specified in QLEV may be exceeded if several messages are created for this queue during a single transaction. If the threshold value has been exceeded, then the behavior will depend on the value set in the operand QMODE=, see below.

With QLEV=0 no messages may be saved in the queue and with QLEV=32767 the queue length is not restricted.

Default: 32767

Minimum value: 0

Maximum value: 32767

If a value is specified that is greater than the maximum, this is set back to the default in the KDCDEF run. No message is issued.

QMODE = **(Queue Mode)**

Determines the behavior of openUTM in the event that the maximum permitted number of messages that may be saved in the USER queue has been reached and thus the queue level (QLEV= operand) has also been reached.

STD When the queue level is reached openUTM rejects all additional messages for the queue with negative return code (40Z for DPUT).

WRAP-AROUND

openUTM continues to accept messages for the queue, even if the queue level has been reached. When a new message is written to the queue, openUTM deletes the oldest message in the queue and replaces it with the new one.

Default: STD

Q-READ-ACL=read-keysetname

Specifies the read and delete rights for external users in the USER queue. In *read-keysetname* you must enter a key set that has been generated using a KSET statement.

If you enter Q-READ-ACL=, an external user (*≠username*) is only permitted read access to the queue if both the key set of their user ID and the key set of the LTERM partner via which the user is signed on contain at least one of the key codes contained in the key set *read-keysetname*.

The owner (*username*) of the USER queue always has read and delete rights to their queue, even if the rights are restricted using Q-READ-ACL.

If you do not specify Q-READ-ACL=, all users have both read and delete rights in the queue.

Default: no key set

Q-WRITE-ACL=write-keysetname

Specifies the write rights for external users in the USER queue.

In *write-keysetname* you must enter a key set that has been generated using a KSET statement.

If you enter Q-WRITE-ACL=, an external user (*≠username*) is only permitted write access to the queue if both the key set of their user ID and the key set of the LTERM partner via which the user is signed on contain at least one of the key codes contained in the key set *write-keysetname*.

The owner (*username*) of the USER queue always has the write rights to their queue, even if the rights are restricted using Q-WRITE-ACL.

If you do not specify Q-WRITE-ACL= all users have both write rights in the queue.

Default: no key set

RESTART= This specifies whether openUTM is to save the service data for a user ID so that a service restart will be possible on the next sign-on under this user ID.

YES The service context belonging to this user ID is saved. This means that a service restart can be performed for users who sign on using this user ID if an open service exists for the user ID.


With a service restart, the type of the client and possibly the generated sign on service may play a role. Additional information can be found in [section “Generating a restart” on page 150](#) and in the openUTM manual “Using openUTM Applications”.

Default: YES

NO The service context belonging to this user ID is not saved, no service restart is possible,

- If the connection is shut down during operation by KDCOFF, if it is lost, or if the application is terminated normally, the service is reset to the last synchronization point and terminated. The event exit VORGANG is then called with KCKNZVG=D (=Disconnect).
- During a UTM warm start following abnormal termination of the application, an open service for this UTM user is terminated without calling the event exit VORGANG.
- Following connection setup, KDCDISP/KDCLAST behaves in the same way as after regeneration.

If RESTART=NO is specified together with SIGNON MULTI-SIGNON=YES, several users can sign on simultaneously to openUTM under this user ID, but only one user can sign on to the terminal. Conversely, it is possible for any number of client programs can sign on simultaneously.

B	SATSEL=	SAT logging mode for this user
B		If SAT logging is activated (MAX SAT=YES), all events triggered by this user are logged as defined in this operand.
B		
B		The SATSEL control statement is used to define the general SAT logging mode for all TACs and users. This can be supplemented by the SATSEL operand of the USER statement, which allows you to define user-specific logging. If the logging of an event class is prohibited in the SATSEL statement, events of this class are not logged. (For information on the link between the EVENT-, TAC- and USER-specific log settings, see the openUTM manual "Using openUTM Applications under BS2000/OSD".)
B		
B		
B		
B		
B		
B		
B		SATSEL can be generated even if SAT logging is deactivated (MAX statement with SECLEV=NO and SAT=OFF). In this case, the statements are not effective when the application is started, but SAT logging is predefined. When required, SAT logging can then be activated during operation (UTM SAT administration command KDCMSAT, see the openUTM manual "Using openUTM Applications under BS2000/OSD").
B	NONE	A user-specific SAT logging mode is not defined.
B		Default: NONE
B	BOTH	Both successful and unsuccessful events are logged.
B	SUCC	Only successful events are logged.
B	FAIL	Only unsuccessful events are logged.
	STATUS=	Status (locked or unlocked) of the user ID when the application is started.
	ON	The user ID is unlocked. Default: ON
	OFF	The user ID is locked. It cannot be used by a user or client to sign on to the application until it has been released by the administrator.
		User IDs that are implicitly or explicitly assigned to an UPIC client or a client of a TS application via an LTERM statement (LTERM ...,USER=) are always locked. They cannot be authorized by the UTM administrator. These user IDs are called connection user IDs.

UTMD - application parameters for distributed processing

The UTMD control statement allows you to define global values for the local UTM application for distributed processing. A UTMD statement is only required for applications that use either the LU6.1 or the OSI TP protocol for communication.

The UTMD statement may only be entered once.

If you use the OSI TP protocol in your application, you can specify the Application Process Title (APT) of the application in the UTMD statement. This is required by some heterogeneous partners that support another variant of the OSI TP protocol in order to establish a connection. These applications expect the specification of the Application Process Title on establishment of the connection.

The application process title is combined with any application entity qualifier (AEQ) assigned to an access point of your application to form an application entity title (AET), which is unique throughout the OSI network. This is used by the partner application to identify the access point of the local application via which communication is to take place.

```

UTMD_      [ APPLICATION-PROCESS-TITLE=object_identifier ]
           [ ,CONCTIME={ time1 | ( time1,time2 ) } ]
           [ ,MAXJR=%_maxjr ]
           [ ,PTCTIME=time3 ]
           [ ,RSET={ GLOBAL | LOCAL } ]

```

APPLICATION-PROCESS-TITLE=object_identifier

(only relevant if the OSI TP protocol is used in the application)

Address component of the application entity title (AET). The AET is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET to establish a connection.

object_identifier is the application process title (APT) of your application. If this is not defined by a standardization body, the relevant conventions for *component1* and *component2* must be observed when assigning an APT. Further information can be found in section “[Application entity title \(AET\)](#)” on page 96. In practice, the specified *object_identifier* must be unique within the network.



If the application context (definition of the communication partner in the OSI-LPAP statement, [page 426](#)) agreed with a partner application contains the CCR syntax, you must enter an application process title here.

An application process title consists of at least 2 and at most 10 components. It is specified in the following format:

(component1,component2,...,component10)

The components are specified in the form of positive integers. Symbolic names are assigned to some numbers of individual components, and can be used instead of the numbers. In the application process title, both the number of components and their positions within the parentheses are relevant, e.g. (1,2,3), (1,2,3,0,0) and (0,1,2,3,0) identify different application process titles.

openUTM and the OSI standard only permit the following values or symbolic names for *component1*:

0 or CCITT

1 or ISO

2 or JOINT-ISO-CCITT

The values permitted for *component2* depend on the value of *component1*.

- If *component1* = 0 or 1, values between 0 and 39 are permitted for *component2* ($0 \leq \text{component2} \leq 39$).
- If *component1* = 2, values between 0 and 67108863 ($2^{26}-1$) are permitted for *component2* ($0 \leq \text{component2} \leq 67108863$).

Values between 0 and 67108863 ($2^{26}-1$) are permitted for all other components.

openUTM does not check whether the specified application process title is registered with a standardization body.

CONCTIME= (**connection control time**)

time1

Maximum number of seconds for which openUTM monitors the opening of a session (LU6.1) or association (OSI TP). If the session or association is not opened within the specified time, openUTM shuts down the transport connection. This prevents the transport connection from being blocked if an attempt to open a session or association fails. This can occur if a message required to open the session/association is lost.

CONCTIME=0 for LU6.1 means opening is not monitored.

CONCTIME=0 for OSI TP means monitoring is set internally to 60 seconds.

Default: 0

Minimum value: 0

Maximum value: 32767

`time2` Maximum number of seconds for which openUTM is to wait for confirmation from the partner application when sending an asynchronous message. Once the specified time has elapsed, openUTM shuts down the transport connection. The job is not lost however. Monitoring prevents the connection from being blocked because a confirmation has been lost, or because the loss of connection was not reported to openUTM by the transport system. The value 0 means that monitoring is not performed.

Default: 0

Minimum value: 0

Maximum value: 32767

`MAXJR=%_maxjr`

(**max**imal number of **j**ob receivers)

Specifies the maximum number of job-receiving services that can be addressed in the local application at any one time.

This corresponds to the number of simultaneously active APRO calls.

The percentage value refers to the number of generated sessions and associations (maximum number of LSES statements for the LU6.1 protocol + total number of parallel connections specified in OSI-LPAP statements; ASSOCIATIONS operand). It must be within the range 0 to 200. If you enter a value > 100, APRO calls issued before the session is reserved can be entered in a table.

Default: 100,

i.e. the maximum number of job-receiving services active at a particular time is equal to the number of sessions and associations.

Minimum value: 0

Maximum value: 200

`PTCTIME=time3`

(**p**repare to **c**ommit)

This is significant only for distributed processing via LU6.1 connections. PTCTIME defines the maximum number of seconds for which a job-receiving service waits in PTC state (transaction status P) for confirmation from the job submitter. Once this time has elapsed, the connection to the job submitter is shut down, the transaction in the job-receiving service is reset, and the service is terminated. This may result in a mismatch. The value 0 means that the job-receiving service waits indefinitely for confirmation.



If a value > 0 is specified in *time3* then this value is ignored by openUTM if a KDCSHUT WARN or GRACE has been issued. In this case, openUTM chooses the wait time in such a way that the transaction is rolled back before the application is terminated in order, if possible, to prevent the application from being terminated abnormally with ENDPET.

- Default:
Value specified in MAX ...,TERMWAIT=*time* for the waiting time after PEND KP
- Minimum value: 0
Maximum value: 32767
- RSET= In the case of Distributed Transaction Processing, this operand defines how resetting a local transaction affects the distributed transaction.
- A local transaction can be reset:
- by a RSET call issued in a program unit, or
 - by resetting a database transaction involved in the local transaction.
- GLOBAL After the local transaction is reset, the program unit must be terminated such that openUTM resets the distributed transaction.
- Default: GLOBAL
- LOCAL Resetting the local transaction has no effect on the distributed transaction.
- Inconsistencies may occur in the distributed databases if some of the local transactions involved in a distributed transaction are reset while others are concluded. If RSET=LOCAL is specified, it is no longer possible to guarantee global data integrity in the relevant system components. This is now the responsibility of the application program units. You must decide when it makes sense to terminate the distributed transaction and when to reset the transaction.

6.6 Dialog control - effects of generation parameters

The following statements and parameters of the KDCDEF generation tool can be used to control the dialog during generation:

KDCDEF statement	Effect
EXIT ...,USAGE=INPUT	Event exit INPUT
LTAC ..., WAITTIME=	Maximum waiting time for distributed processing
LTERM ..., RESTART=	Refers to asynchronous messages and the service restart procedure if user IDs are not generated
LTERM ..., USER=	Automatic KDCSIGN
MAX ..., APPLIMODE=	Refers to the service restart procedure and asynchronous messages
MAX ..., CONN-USERS=	Application load: number of simultaneously active users or clients
MAX ..., NRCONV=	Maximum number of stacked services
MAX ..., PGWTIME=	Maximum time in seconds that a program unit is allowed to wait to receive messages after a blocked call.
MAX ..., TERMWAIT=	Maximum waiting time until the next input from the terminal in a multi-step transaction (following PEND KP)
PTERM ..., IDLETIME= TPOOL ..., IDLETIME=	Maximum waiting time until the next input from the client after the end of the transaction or after signing on (following PEND RE/FI/ER)
SIGNON ..., GRACE=	The user may or may not change his or her password after it has expired
SIGNON ..., MULTI-SIGNON=	Several users/clients may or may not be signed on at the same time under the same user ID
SIGNON ..., SILENT-ALARM=	Limits the number of unsuccessful sign-on attempts
USER statements defined	Sign-on check performed for all users
USER ..., PASS=	Sign-on check with a password; input may be blanked-out
USER ..., PROTECT-PW=	Authorization check with a blanked-out password, validity period, and level of complexity
USER ..., RESTART=	Service restart procedure
SFUNC ...	Assignment of a F key or K key (under BS2000/OSD) as a TAC, UTM command, or stacking request
TAC KDCBADTC	Event service BADTACS
TAC KDCSGNTC	Sign on using a sign-on procedure

	KDCDEF statement	Effect
B	<i>Under BS2000/OSD the following parameters also are relevant</i>	
B	LTERM ..., ANNOAMSG=	Asynchronous messages with or without prior announcement
B	LTERM ..., FORMAT=	LTERM specific start format
B B	TPOOL ..., ANNOAMSG=	Asynchronous messages announced in advance before being output in the system line on the terminal
B B	PTERM ..., CONNECT=Y	Automatic connection setup to the terminal by openUTM when starting the application
B	MAX ..., LOCALE=	Default language environment of the application
B	LTERM ..., LOCALE=	Language environment of clients that sign on via LTERM partners
B	TPOOL ..., FORMAT=	defines the start format for the terminal user
B	TPOOL ..., LOCALE=	Language environment of clients that sign on via LTERM pools
B	USER ..., CARD=	Sign-on check with a magnetic strip card
B B	USER ..., CERTIFICATE=	Specifies if a user must identify himself with a chip card containing the generated certificate when signing on to the UTM application.
B	USER ..., FORMAT=	User specific start format
B	USER ..., LOCALE=	Language environment of the user
B	USER ..., PRINCIPAL=	User authentication will be performed using Kerberos.

6.7 Example generation: *Comfo*TRAVEL

The example generation *Comfo*TRAVEL is a travel reservation system which allows the customers of travel agents in Munich, Paris and New York, to book “all-inclusive” packages consisting of hotel, flight and leisure activities plus the necessary bank operations. To do this, the travel agents access a central reservation system (RMS Reservation Management System).

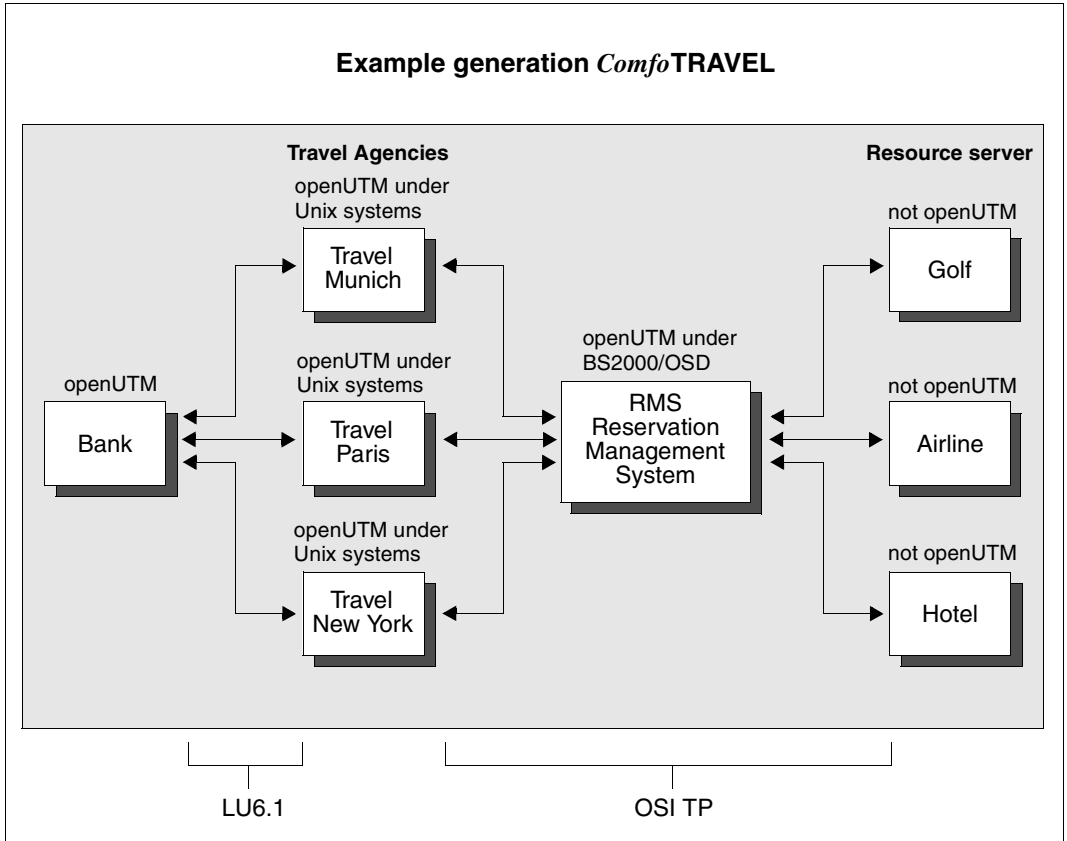


Figure 19: Example generation *Comfo*TRAVEL with the employed protocols

6.7.1 KDCDEF input file DYNAMIC.RMS for UTM-D application RMS

```

*****
*      ADMINISTRATION programs      *
*****

PROGRAM KDCADM, COMP=ILCS
PROGRAM KDCDADM,COMP=ILCS
PROGRAM KDCPADM,COMP=ILCS

*****
*      RMS programs                  *
*****

PROGRAM AVALRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM RESRRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM CNCLRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM AUTHRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM INITRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM SHUTRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM ENQRRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM HNDLEXIT, COMP=ILCS, LOAD-MODULE=RMS

*****
*      BOOKKEEPING programs         *
*****

PROGRAM BOOKKEEP, COMP=ILCS, LOAD-MODULE=BOOKKEEP

*****
*      PERSONNEL programs           *
*****

PROGRAM PERSNNL, COMP=ILCS, LOAD-MODULE=PERSNNL

*****
*      OFFICE programs              *
*****

PROGRAM OFFICE, COMP=ILCS, LOAD-MODULE=OFFICE

*****
*** TAC statements ***
*****

***** TAC BOOKKEEP      *****
*
TAC BOOKKEEP, PROGRAM=BOOKKEEP, LOCK=7

```

```
***** TACS PERSNNL      *****
*
TAC OPERSNNL, PROGRAM=PERSNNL, LOCK=2
TAC MPERSNNL, PROGRAM=PERSNNL, LOCK=3
TAC CPERSNNL, PROGRAM=PERSNNL, LOCK=4

***** TACS OFFICE      *****
*
TAC OFFCHRG , PROGRAM=OFFICE , LOCK=5
TAC OFFADMIN, PROGRAM=OFFICE , LOCK=6
*

***** TACS RMS          *****
*
TAC AVALRESP, PROGRAM=AVALRESP, LOCK=8
TAC RESRRESP, PROGRAM=RESRRESP, LOCK=8
TAC CNCLRESP, PROGRAM=CNCLRESP, LOCK=8
TAC AUTHRESP, PROGRAM=AUTHRESP, LOCK=8
TAC INITRESP, PROGRAM=INITRESP, LOCK=8
TAC SHUTRESP, PROGRAM=SHUTRESP, LOCK=8
TAC ENQRRESP, PROGRAM=ENQRRESP, LOCK=8

*** ADMINISTRATION DIALOG ***

DEFAULT TAC ADMIN=Y, PROGRAM=KDCADM

TAC KDCTAC , LOCK=1
TAC KDCLTERM, LOCK=1
TAC KDCPTERM, LOCK=1
TAC KDCSWTCH, LOCK=1
TAC KDCSEND , LOCK=1
TAC KDCAPPL , LOCK=1
TAC KDCUSER , LOCK=1
TAC KDCDIAG , LOCK=1
TAC KDCLOG , LOCK=1
TAC KDCINF , LOCK=1
TAC KDCHelp , LOCK=1
TAC KDCLPAP , LOCK=1
TAC KDCLTAC , LOCK=1
TAC KDCSHUT , LOCK=1
TAC KDCTCL , LOCK=1

TAC TACDADM , PROGRAM=KDCDADM, LOCK=1
TAC TACPADM , PROGRAM=KDCPADM, LOCK=1
```


*** ADMINISTRATION ASYNCHRON ***

DEFAULT TAC TYPE=A

TAC KDCTACA , LOCK=1
 TAC KDCLTRMA, LOCK=1
 TAC KDCPTRMA, LOCK=1
 TAC KDCSWCHA, LOCK=1
 TAC KDCUSERA, LOCK=1
 TAC KDCSEDA, LOCK=1
 TAC KDCAPPLA, LOCK=1
 TAC KDCDIAGA, LOCK=1
 TAC KDCLOGA , LOCK=1
 TAC KDCINFA , LOCK=1
 TAC KDCHELPA, LOCK=1
 TAC KDCLPAPA, LOCK=1
 TAC KDCLTACA, LOCK=1
 TAC KDCSHUTA, LOCK=1
 TAC KDCTCLA , LOCK=1

*
 *
 *
 *
 *

 *** USER statements ***

USER SUPERUSR, PERMIT=ADMIN, PASS='\$23ADM--', PROTECT-PW=(8, MAX) -
 , KSET=MASTER
 USER UTMADMIN, PERMIT=ADMIN, PASS='\$23ADM--', PROTECT-PW=(8, MAX) -
 , KSET=UTMADMIN
 USER CLERK , FORMAT=*BOOK, PASS='a\$*45jk1', PROTECT-PW=(8, MAX) -
 , KSET=BOOKKEEP
 USER PRSNLMNG, FORMAT=*PERSNNL, PASS='78+1sd*/', PROTECT-PW=(8, MAX) -
 , KSET=MPRSNNL
 USER MILLER , FORMAT=*PERSNNL, PASS='7HGFKK*/', PROTECT-PW=(, MED) -
 , KSET=OPRSNNL
 USER COMP , FORMAT=*PERSNNL, PASS='7sdfKK*/', PROTECT-PW=(, MED) -
 , KSET=CPRSNNL
 USER CHARGE , FORMAT=*TRAVEL, PASS='%aJ1df-+', PROTECT-PW=(, MED) -
 , KSET=OFFCHRG , LOCALE=(EN)
 USER CHIEF , FORMAT=*TRAVEL, PASS='%aJs5f-+', PROTECT-PW=(, MED) -
 , KSET=OFFADMIN, LOCALE=(EN)
 USER TPARIS , FORMAT=*TRAVEL, PASS='kj678+*', PROTECT-PW=(, MED) -
 , KSET=TRVAGNCY, LOCALE=(FR, EU)
 USER TNEWYORK, FORMAT=*TRAVEL, PASS='56asdf\$~', PROTECT-PW=(, MED) -
 , KSET=TRVAGNCY, LOCALE=(EN)

```

USER TMUNICH , FORMAT=*TRAVEL, PASS='%as3f$0', PROTECT-PW=( , MED) -
, KSET=TRVAGNCY, LOCALE=(DE, EU)
USER TLONDON , FORMAT=*TRAVEL, PASS='%4Jsdf-+', PROTECT-PW=( , MED) -
, KSET=TRVAGNCY, LOCALE=(EN)
USER MANOFF , FORMAT=*BOOK, PASS='$23ADM--', PROTECT-PW=(8, MAX) -
, KSET=OFFADMIN

*****
*** PTERM/LTERM statements ***
*****

PTERM PRB22273, LTERM=PRINTER, PRONAM=PRO, PTYPE=T9021, CONNECT=YES
LTERM PRINTER, USAGE=0

PTERM RSO, LTERM=RSO, PTYPE=*RSO, PRONAM=*RSO, CONNECT=YES
LTERM RSO, USAGE=0

```

6.7.2 KDCDEF statements for UTM-D application RMS

```

*****
*** K D C D E F - S T A T E M E N T S ***
*** FOR UTM-D-PROGRAM "RMS" ***
*****

ROOT RMSROOT

OPTION GEN=ALL

ACCOUNT ACC = YES

FORMSYS

MESSAGE MODULE = KCSMSGs, LOCALE=(EN)
MESSAGE MODULE = MSGSGER, LOCALE=(DE, EU)
MESSAGE MODULE = MSGSFRA, LOCALE=(FR, EU)

MAX LOCALE = (EN)

MAX KDCFILE = (RMS, DOUBLE) -
,APPLNAME = APRMS -
,APPLMODE = S -
,TASKS = 10 -
,ASYNTASKS = 3 -
,GSSBS = 200 -
,PGPOOL = 2048 -
,CACHESIZE = (512,50,RES) -
,CONN-USERS = 50 -
,RECBUF = (10,1024) -
,KEYVALUE = 20 -
,LSSBS = 9 -
,LPUTBUF = 10 -

```

```

,LPUTLTH = 1948      -
,NRCONV = 1         -
,TERMWAIT = (600)   -
,DPUTLIMIT1 = (363,0,0,0) -
,DPUTLIMIT2 = (1,0,0,0) -
,KB = 1024          -
,NB = 2048          -
,SPAB = 4096        -
,CLRCH = X'FF'

```

 *** RESERVE statement to allow dynamic administration ***

```

RESERVE OBJECT=ALL

```

 *** DATABASE CONTROL statements ***

```

DATABASE TYPE=UDS
DATABASE TYPE=ORACLE

```

*
 *

 *** SFUNC CONTROL statements ***

```

SFUNC F1 , TAC = INITRESP
SFUNC F2 , TAC = SHUTRESP
SFUNC F5 , TAC = ENQRRESP
SFUNC F6 , TAC = AUTHRESP
SFUNC F7 , TAC = RESRRESP
SFUNC F8 , TAC = AVALRESP
SFUNC F20, TAC = CNCLRESP

```

 *** KSET statements ***

```

KSET MASTER , KEYS=MASTER      "SUPERUSER"
KSET UTMADMIN, KEYS=1          "Administrator of application"
KSET OPRSNL , KEYS=2           "office personnel / Büropersonal"
KSET MPRSNL , KEYS=3           "personnel manager / Personalchef"
KSET CPRSNL , KEYS=4           "computer personnel / DV-Mitarbeiter"
KSET OFFCHRG , KEYS=5          "official in charge / Sachbearbeiter"
KSET OFFADMIN, KEYS=6          "administrator of office data"
KSET BOOKKEEP, KEYS=7          "book keeper"
KSET TRVAGNCY, KEYS=8          "travel agencies"

```

```

*****
*** LOAD-MODULE statements ***
*****

LOAD-MODULE BOOKKEEP, VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
LOAD-MODULE PERSNNL , VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
LOAD-MODULE RMS      , VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
LOAD-MODULE OFFICE   , VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP

*****
*** EXIT statements ***
*****

EXIT PROGRAM=HNDLEXIT, USAGE=START
EXIT PROGRAM=HNDLEXIT, USAGE=SHUT

*****
*** Read data which could be administered dynamically ***
*****

* use create-control-statements if application ran before
* CREATE-CONTROL-STATEMENTS *ALL, TO-FILE   = DYNAMIC.RMS.DATA -
*                                     , FROM-FILE = COPIED.RMS.KDCA

OPTION DATA=DYNAMIC.RMS

*
*
*
*
*
*

*****
*** TACCLASS statements ***
*****

* not used

*****
*** TLS statements ***
*****

TLS TLSA

*****
*** ULS statements ***
*****

ULS ULSA
ULS ULSB

```

```

*****
*** TPOOL statements                                     ***
*****

TPOOL LTERM=TP#,   NUMBER=100, PRONAM=*ANY, PTYPE=*ANY, KSET=MASTER
TPOOL LTERM=UPICR, NUMBER=100, PRONAM=*ANY, PTYPE=UPIC-R, KSET=MASTER

*****
*** UTMD statements                                     ***
*****

UTMD MAXJR = 200, APT=(1,2,3,10), CONCTIME=25, PTCTIME=0

*****
*** Generation of syntax                               ***
*****

ABSTRACT-SYNTAX EUROSI, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12) -
      , TRANSFER-SYNTAX = BER

*****
* Generation of APPLICATION CONTEXTS                 ***
*****
*
* Without CCR
*
APPLICATION-CONTEXT EUROSIAC, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 12) -
      , ABSTRACT-SYNTAX = (EUROSI)
*
* Include CCR
*
APPLICATION-CONTEXT EUOSICCR, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13) -
      , ABSTRACT-SYNTAX = (EUROSI, CCR)
*
*
*
*
*
*
*

```

```

*****
*** OSI TP generation ***
*****

*+-----+
*|                                     |
*|          T R A V E L  - Connections |
*|                                     |
*+-----+

ACCESS-POINT RMS, T-SEL=C'RMS', S-SEL=('SRMS',ASCII) -
          , P-SEL=('PRMS',ASCII), AEQ=1
*
*
*   travel-agency MUNICH <=====> RMS
OSI-CON MUNICH, LOCAL-ACCESS-POINT=RMS, OSI-LPAP=MUNICH -
          , N-SEL=C'HOST0001', T-SEL=C'TRAV', S-SEL=(C'STRV',ASCII) -
          , P-SEL=(C'PTRV',ASCII)
*
*   travel-agency PARIS <=====> RMS
OSI-CON PARIS , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=PARIS -
          , N-SEL=C'ISO09', T-SEL=C'TRAV', S-SEL=(C'STRV',ASCII) -
          , P-SEL=(C'PTRV',ASCII)
*
*   travel-agency NEWYORK <=====> RMS
OSI-CON NEWYORK, LOCAL-ACCESS-POINT=RMS, OSI-LPAP=NEWYORK -
          , N-SEL=C'ISO10', T-SEL=C'TRAV', S-SEL=('2',ASCII) -
          , P-SEL=('2',ASCII)
*
*   travel-agency LONDON <=====> RMS
OSI-CON LONDON , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=LONDON -
          , N-SEL=C'ISO06', T-SEL=C'TRAV', S-SEL=('2',ASCII) -
          , P-SEL=('2',ASCII)
*
OSI-LPAP MUNICH , ASS-NAMES=MUNICH, ASSOCIATIONS=4, CONNECT=0 -
          , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR -
          , APT=(1,2,3,21),AEQ=1, KSET=TRVAGNCY
OSI-LPAP PARIS , ASS-NAMES=PARIS, ASSOCIATIONS=4, CONNECT=0 -
          , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR -
          , APT=(1,2,3,22), AEQ=1, KSET=TRVAGNCY
OSI-LPAP NEWYORK, ASS-NAMES=NEWYORK, ASSOCIATIONS=1, CONNECT=0 -
          , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR -
          , APT=(1,2,3,23), AEQ=1, KSET=TRVAGNCY
OSI-LPAP LONDON , ASS-NAMES=LONDON, ASSOCIATIONS=1, CONNECT=0 -
          , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR -
          , APT=(1,2,3,24), AEQ=1, KSET=TRVAGNCY
*
*

```

```

*
*
*
*
*
*
*
*
*
*+-----+
*|          From RMS to all servers          |
*+-----+
*
*
*      RMS  <=====> Server
*
OSI-LPAP BANK    , ASS-NAMES=BANK, ASSOCIATIONS=4, CONNECT=4  -
                  , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR  -
                  , APT=(1,2,3,30), AEQ=1
OSI-LPAP GOLF    , ASS-NAMES=GOLF, ASSOCIATIONS=4, CONNECT=4  -
                  , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR  -
                  , APT=(1,2,3,30), AEQ=2
OSI-LPAP HOTEL   , ASS-NAMES=HOTEL, ASSOCIATIONS=4, CONNECT=4  -
                  , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR  -
                  , APT=(1,2,3,30), AEQ=3
OSI-LPAP AIRLINE, ASS-NAMES=FLIGHT, ASSOCIATIONS=4, CONNECT=4  -
                  , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR  -
                  , APT=(1,2,3,30), AEQ=4
*
*
LTAC BANK, LPAP=BANK,      RTAC=BANK, STATUS=ON, TYPE=D
*
*
OSI-CON BANK     , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=BANK      -
                  , N-SEL=C'HOST0001', T-SEL=C'BANK', S-SEL=('SBNK',ASCII) -
                  , P-SEL=(C'PBNK',ASCII)
OSI-CON GOLF     , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=GOLF      -
                  , N-SEL=C'HOST0001', T-SEL=C'GOLF', S-SEL=('SGLF',ASCII) -
                  , P-SEL=(C'PGLF',ASCII)
OSI-CON HOTEL    , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=HOTEL     -
                  , N-SEL=C'HOST0001', T-SEL = C'HOTL'      -
                  , S-SEL = ('SHTL',ASCII), P-SEL = ('PHTL',ASCII)
OSI-CON AIRLINE  , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=AIRLINE   -
                  , N-SEL=C'HOST0001', T-SEL = C'FLGH'      -
                  , S-SEL=(C'SFLG',ASCII), P-SEL=(C'PFLG',ASCII)

END

```

6.7.3 KDCDEF input file DynamicTravel for UTM application TRAVEL

```

*****
*   BANK program                               *
*****

PROGRAM BANK,      COMP=C, SHARED-OBJECT=BANK

*****
*   TRAVEL programs                            *
*****

PROGRAM SIGN1,     COMP=C
PROGRAM SIGN2,     COMP=C
PROGRAM BDTAC,     COMP=C
PROGRAM TRRECEIV,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM STRTEX,    COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM MMENUE,    COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO1,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO2,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO3,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO4,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO5,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO6,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM CANCEL,    COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM CANCELL,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINQU,    COMP=C, SHARED-OBJECT=TRAVEL

**** Administration

PROGRAM KCADM,     COMP=C
PROGRAM KCDADM,    COMP=C
PROGRAM KDCPADM,   COMP=C

*****
*   TACS BANK                                           *
*****

TAC BANK, PROGRAM=BANK, LOCK=5

*****
*   TACS TRAVEL AGENCY                                  *
*****

TAC KDCBADTC, PROGRAM=BDTAC, TYPE=D
TAC KDCSGNTC, PROGRAM=SIGN1, TYPE=D
TAC SIGNON2 , PROGRAM=SIGN2, TYPE=D
TAC MMENUE , PROGRAM=MMENUE , LOCK=5
TAC INFO1 , PROGRAM=TRINFO1 , LOCK=5
TAC INFO2 , PROGRAM=TRINFO2 , LOCK=5
TAC INFO3 , PROGRAM=TRINFO3 , LOCK=5
TAC INFO4 , PROGRAM=TRINFO4 , LOCK=5

```



```

TAC INFO5 , PROGRAM=TRINFO5 , LOCK=5
TAC INFO6 , PROGRAM=TRINFO6 , LOCK=5
TAC TRRECEIV, PROGRAM=TRRECEIV, LOCK=5
TAC CANCEL , PROGRAM=CANCEL , LOCK=5, CALL=NEXT, TYPE=D
TAC CANCELL , PROGRAM=CANCELL , LOCK=5, CALL=FIRST, TYPE=D
TAC INQUIRY , PROGRAM=TRINQU , LOCK=5
*
```

```

**** ADMINISTRATION DIALOG ***
```

```

TAC KDCTAC , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLTERM, LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCPTERM, LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCSWTCH, LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCSEND , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCAPPL , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCUSER , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCDIAG , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLOG , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCINF , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCHELP , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLPAP , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLTAC , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCSHUT , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCTCL , LOCK=1, ADMIN=Y, PROGRAM=KDCADM

TAC TACDADM , PROGRAM=KDCDADM, LOCK=1, ADMIN=Y
TAC TACPADM , PROGRAM=KDCPADM, LOCK=1, ADMIN=Y
```

```

*** ADMINISTRATION ASYNCHRON ***
```

```

TAC KDCTACA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLTRMA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCPTRMA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCSWCHA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCUSERA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCSEDA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCAPPLA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCDIAGA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLOGA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCINFA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCHELPA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLPAPA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLTACA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCSHUTA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCTCLA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
*
```

```

*****
*** USER statements                                     ***
*****

USER SUPERUSR, PERMIT=ADMIN,      PASS='$23ADM--', PROTECT-PW=(8, MAX)      -
, KSET=MASTER
USER UTMADMIN, PERMIT=ADMIN,      PASS='$23ADM--', PROTECT-PW=(8, MAX)      -
, KSET=UTMADMIN
USER CHARGE1 , FORMAT=(TRAVEL, EXTEND),PASS='%aJ1df+', PROTECT-PW=( ,MED) -
, KSET=OFFCHRG
USER CHARGE2 , FORMAT=(TRAVEL, EXTEND),PASS='%aJ1df+', PROTECT-PW=( , MED) -
, KSET=OFFCHRG
*
.
*****
*** PTERM/LTERM statements                             ***
*****

PTERM PRINTX, LTERM=PRINTER, PTYPE=PRINTER, CONNECT=YES
LTERM PRINTER, USAGE=0

```

6.7.4 KDCDEF statements for UTM application TRAVEL

```

*****
*** K D C D E F - S T A T E M E N T S                 ***
*** FOR UTM-PROGRAM "TRAVEL"                          ***
*****

ROOT TRAVROOT

OPTION GEN=ALL

FORMSYS

MESSAGE MODULE = KCSMSGs

MAX KDCFILE = (TRAVFILE, DOUBLE) -
,APPLNAME = APTRAVEL             -
,APPLMODE = S                    -
,TASKS = 7                       -
,ASYNTASKS = 3                   -
,GSSBS = 200                     -
,PGPOOL = (2048)                 -
,CACHESIZE = (512,50)            -
,CONN-USERS = 50                 -
,TRACEREC = 30000                -
,RECBUF = (10,1024)              -
,KEYVALUE = 20                   -
,LSSBS = 9                       -
,LPUTBUF = 10                    -

```

```

,LPUTLTH = 1948          -
,NRCONV = 1             -
,TERMWAIT = (600)      -
,DPUTLIMIT1 = (363,0,0,0) -
,DPUTLIMIT2 = (1,0,0,0) -
,KB = 1024             -
,NB = 2048            -
,SPAB = 4096          -
,CLRCH = X'FF'        -
,SEMARRAY   =(00001221,5) -
,IPCSHMKEY  = 00012210   -
,KAASHMKEY  = 00012220   -
,CACHESHMKEY = 00012230   -
,OSISHMKEY  = 00012244   -
,XAPTPSHMKEY = 00012254

*****
*** Read data which can be administrated dynamically ***
*****

* if application ran before use create-control-statements
* CREATE-CONTROL-STATEMENTS *ALL, TO-FILE   = dynamicTravel   -
*                                     , FROM-FILE = TRAVFILE/copied.KDCA
OPTION DATA=DynamicTravel
*
*

*****
*** RESERVE statement to allow dynamic administration ***
*****

RESERVE OBJECT=ALL

*****
***          RMXA                                     ***
*****

RMXA XASWITCH=xaswitch

*****
*** SHARED-OBJECT statements                               ***
*****

SHARED-OBJECT TRAVEL, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
SHARED-OBJECT BANK,   LIB=DYNPROGLIB, LOAD-MODE=STARTUP

```

```

*****
*** KSET statements                                     ***
*****

KSET MASTER , KEYS=MASTER           "SUPERUSER"
KSET UTMADMIN, KEYS=1                "Administrator of application"
KSET OFFCHRG , KEYS=5                "official in charge / Sachbearbeiter"

*****
*** TPOOL statements                                   ***
*****

TPOOL LTERM=TP#, NUMBER=100, PTYPE=TTY, KSET=MASTER
TPOOL LTERM=UPICR, NUMBER=100, PTYPE=UPIC-R, KSET=MASTER

*****
*** Generation of syntax                               ***
*****

ABSTRACT-SYNTAX EUROSI, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12) -
, TRANSFER-SYNTAX = BER

*****
* Generation of APPLICATION CONTEXTS                 ***
*****
*
* Without CCR
*
APPLICATION-CONTEXT EUROSIAC, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 12) -
, ABSTRACT-SYNTAX = (EUROSI)

*
* Include CCR
*
APPLICATION-CONTEXT EUOSICCR, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13) -
, ABSTRACT-SYNTAX = (EUROSI, CCR)

*
*
*
*****
*** OSI TP generation                                 ***
*****

*****
*** UTMd statements                                   ***
*****

UTMD MAXJR = 200, APT=(1,2,3,21), CONCTIME=25, PTCTIME=0

```

```

*+-----+
*|                                     |
*|               R M S - Connections |
*|                                     |
*+-----+

ACCESS-POINT TRAVEL, T-SEL=C'TRAV', S-SEL= (C'STRV',ASCII) -
                , P-SEL= (C'PTRV',ASCII), AEQ=1 -
                , LISTENER-PORT=30003, T-PROT=RFC1006 , TSEL-FORMAT=T

*+-----+
*|               From travel agency to RMS |
*+-----+
*
*   travel-agency <=====> RMS
OSI-CON RMS , LOCAL-ACCESS-POINT=TRAVEL, OSI-LPAP=RMS,N-SEL=C'HOST0001'-
                ,T-SEL=C'RMS',S-SEL= (C'SRMS',ASCII), P-SEL= (C'PRMS',ASCII)-
                ,LISTENER-PORT=102, T-PROT=RFC1006, TSEL-FORMAT=T
OSI-LPAP RMS, ASS-NAMES=RMS, ASSOCIATIONS=4, CONNECT=0, CONTWIN=4      -
                , APPLICATION-CONTEXT=EUOSICCR, APT=(1,2,3,10),AEQ=1

*
*
*+-----+
*|                                     |
*|               B A N K - Connections |
*|                                     |
*+-----+

SESCHA PLUC, PLU=Y, PACCNT=0, CONNECT=Y

LPAP LPBANK, SESCHA=PLUC

BCAMAPPL SMP30041                                     -
                ,T-PROT=RFC1006                                     -
                ,LISTENER-PORT=30004,TSEL-FORMAT=T
* Connection 1 for sending ---> BANK-----*
CON SMP30114,PRONAM=local,BCAMAPPL=SMP30041,LPAP=LPBANK -
                ,T-PROT=RFC1006                                     -
                ,LISTENER-PORT=30001,TSEL-FORMAT=T
LSES SMP30141,RSES=SMP30141,LPAP=LPBANK
* Connection 2 for sending ---> BANK-----*
CON SMP30214,PRONAM=local,BCAMAPPL=SMP30041,LPAP=LPBANK -
                ,T-PROT=RFC1006                                     -
                ,LISTENER-PORT=30001,TSEL-FORMAT=T
LSES SMP30241 ,RSES=SMP30241,LPAP=LPBANK

```

```
* Connection 3 for sending ----> BANK-----*
CON SMP30314,PRONAM=local,BCAMAPPL=SMP30041,LPAP=LPBANK -
      ,T-PROT=RFC1006 -
      ,LISTENER-PORT=30001,TSEL-FORMAT=T
LSES SMP30341,RSES=SMP30341,LPAP=LPBANK

*-----*
*          LTAC's          -----> BANK
*-----*

LTAC bank, RTAC=BANK, WAITTIME=(10,30), LPAP=LPBANK
*-----*
*          LTAC's          -----> RMS
*-----*

LTAC AVALRESP, LPAP=RMS
LTAC RESRRESP, LPAP=RMS
LTAC CNCLRESP, LPAP=RMS
LTAC AUTHRESP, LPAP=RMS
LTAC INITRESP, LPAP=RMS
LTAC SHUTRESP, LPAP=RMS
LTAC ENQRRESP, LPAP=RMS

*

END
```

6.8 KDCDEF messages

The KDCDEF generation tool logs the defined parameters on SYSLST (BS2000/OSD) or on *stdout* (Unix systems and Windows systems). It also outputs UTM messages relating to execution of the program, with message numbers ranging from K400 to K549. Apart from UTM messages K401, K513 and K514 all KDCDEF messages are output both to SYSLST and to SYSOUT or to *stderr* and to *stdout*. UTM messages K401, K513 and K514 is output only to SYSOUT or *stderr*.

Under Unix systems KDCDEF uses NLS message catalogs to output messages.

UTM messages relating to incorrect statements are accompanied by the relevant statement. The openUTM manual “Messages, Debugging and Diagnostics” lists all UTM messages together with information on the corrective actions to be performed.

7 Changing the configuration of an application dynamically

This chapter describes what to note in the KDCDEF generation of the application if you want to use the dynamic configuration functions in your application. On the program interface, openUTM provides KDCADMI functions as well as functions available at the administration workstation WinAdmin with which you can enter objects in the configuration of the application or delete them from the configuration while the application is running. This increases the availability of UTM applications, because a regeneration of the application with KDCDEF, which necessitates an interruption to operation, is required much less often. In order to use the functions for dynamic configuration, you must reserve table locations in the object tables of openUTM when generating with the KDCDEF control statement RESERVE.

This means that services as well as clients and printers can be entered dynamically in the configuration with the assigned LTERM partners, and also means that user IDs can be created dynamically. All of these objects can also be deleted dynamically.

You can dynamically create and delete the following objects:

- transaction codes and TAC queues
- program units and VORGANG exits (only in applications with load modules, shared objects or DLLs)
- user IDs
- LTERM partners
- key sets
- local service names
- transport connections to LU6.1 partner applications and LU6.1 session names
- communication partners that are TS applications, UPIC clients or terminals
- printers.

B/X

To be able to use the functions of dynamic configuration, you must create administration programs or use the openUTM component “openUTM WinAdmin”. By calling `KC_CREATE_OBJECT` on the program interface for administration you can enter new objects in the configuration, and by calling `KC_DELETE_OBJECT` you can delete objects from the configuration. The openUTM manual “Administering Applications” describes what to note when creating administration programs for dynamically entering objects and when deleting objects from the configuration of the application.



The dynamic configuration functions can also be used in full in the function variant UTM-F. openUTM logs all changes to the configuration in the `KDCFILE`. The modified configuration data then also remains available for the next application run, as with UTM-S.

To allow you to incorporate objects into the configuration of your UTM application dynamically, you must make certain preparations (see [page 571](#) and [page 573](#)) when generating the application with `KDCDEF`.

No preparations are necessary in the `KDCDEF` generation for deleting objects from the configuration.

7.1 Reserving locations in the KDCFILE object tables

The configuration data of a UTM application is stored in the object tables of the KDCFILE, which is created in the KDCDEF generation of the application. These object tables can only be expanded dynamically insofar as free table locations are available. For this reason, free table locations for objects you do not want to incorporate into the application configuration until the application is running, must still be reserved when generating with the KDCDEF statement RESERVE. You can reserve table locations for the following UTM objects:

UTM object	Object type
User IDs	USER
TS applications, UPIC clients, terminals and printers	PTERM
LTERM partners	LTERM
Program units and VORGANG exits	PROGRAM
Transaction codes and TAC queues	TAC
Transport connections to LU6.1 partner applications	CON
LU6.1 session names	LSES
Key sets	KSET
Local service names	LTAC

With the RESERVE statement on [page 461](#), you define the number of empty table locations to be created for an object type; this corresponds to the number of individual objects of the respective object type that can be configured dynamically.

The number of table locations that can be created for each object type is limited by the number of names that can be generated. See also the table on [page 263](#).

The empty table locations in the object tables are reserved for specific object types, i.e. a table location you reserved for an LTERM partner for example, cannot be occupied by a transaction code, etc.

During the application run, the number of objects of a particular type that can be configured dynamically corresponds to the number of empty table locations you reserved with KDCDEF. Deleting another object of the same type does not immediately release a table location for a new object.

User IDs are one exception to this. You can delete user IDs in one of two ways, "delayed" or "immediately". If a user ID is deleted with "delayed", then the table locations remain reserved (as for the other types of objects). If the user ID is removed "immediately" from the configuration, the table location for this user ID is released and can be used immediately for a new user ID.



Please note the following when reserving table locations with RESERVE: openUTM internally creates a user ID for each UPIC client and each TS application entered dynamically in the configuration. Therefore, in UTM applications generated with user IDs (the KDCDEF generation contains at least one USER statement), an additional table location for user IDs must be reserved for each client of type APPLI, SOCKET, UPIC-R or UPIC-L to be entered dynamically. These table locations are not released when the clients are deleted (corresponds to a “delayed” delete). In applications without user IDs, these table locations are reserved internally by openUTM.

Examples

```
RESERVE OBJECT=LTERM, NUMBER=100
```

This means that up to 100 LTERM partners can be entered dynamically in the configuration.

```
RESERVE OBJECT=LTERM, PERCENT=200
```

In this case, the number of reserved table locations was defined relative to the number of statically generated LTERM partners. Twice as many (200%) LTERM partners can be created dynamically as were entered statically in the KDCDEF generation. If 50 LTERM partners were entered in the KDCDEF generation, another 100 LTERM partners can be entered dynamically.

```
RESERVE OBJECT=ALL, NUMBER=100
```

This means that 100 objects can be entered dynamically for each object type, i.e. 100 user IDs, 100 LTERM partners, etc.

```
RESERVE OBJECT=USER, NUMBER=0
```

This statement means that the number of objects of the specified type (here USER) can be increased dynamically up to the maximum value that can be generated.



Due to the large amount of space required by the tables, it is advisable to specify a value $\neq 0$ for NUMBER in order to reduce the space requirement of the application.

7.2 Prerequisites for entering objects dynamically

This section describes what objects you must generate statically beforehand and what prerequisites must be met before you can dynamically enter program units, VORGANG exits, transaction codes, user IDs and LU6.1 connections.

Note that the statically generated limit values also apply for dynamically generated objects, e.g. the value defined with `MAX ...,KEYVALUE=` applies to dynamically generated key sets.

Generating lock codes, BCAMAPPL names, formatting system and LPAP partners

The following objects must be statically generated in KDCDEF:

- The lock codes you want to assign to transaction codes and LTERM partners as data access control must lie within the range between 1 and the maximum value defined with `MAX, ...,KEYVALUE= number`. At the same time, you must generate the key sets that contain the key codes corresponding to the lock codes.

The lock/key code concept is described in detail in the openUTM manual “Concepts und Functions”.

- All names of the local application (BCAMAPPL names) via which the connections are to be established to clients or printers, must be generated with KDCDEF. Remember especially that a separate BCAMAPPL name must be generated for the connection of TS applications via the socket interface (`PTYPE=SOCKET`).
- If start formats are to be assigned to user IDs and LTERM partners, a format handling system must be generated with the FORMSYS statement in the KDCDEF generation. If #formats are used as start formats, a sign-on service must also be generated.
- If you want to enter LU6.1 connections or session names dynamically, the LPAP partners and the session characteristics (SESCHA statement) must be statically generated.

B
B
B

Prerequisites for program units and VORGANG exits

New program units and VORGANG exits can only be incorporated dynamically into the configuration of the application if the UTM application

- B** ● *BS2000/OSD:*

B The application was generated with load modules (KDCDEF generation with LOAD-MODULE statements), and the functionality of the BLS must be used for linking and loading the application program.

B

B The new program unit must be linked in a load module which was defined in the KDCDEF generation. This load module must not be linked statically in the application program (LOAD-MODE=STATIC), because this type of load module cannot be exchanged dynamically.


B
- X/W** ● *Unix systems, Windows systems:*

X/W The application was generated with shared objects or DLLs (KDCDEF generation with SHARED-OBJECTS statements).

X/W The new program unit must be linked in a shared object or DLL which was defined in the KDCDEF generation.

X/W

At least one program unit must be generated statically with KDCDEF for each programming language in which you want to create program units of your application. Only then the language connection modules and runtime systems are required for operation contained in the application program.

- B**  *BS2000/OSD:*

B For program units compiled with ILCS-capable compilers (COMP=ILCS), it is sufficient to generate one program unit with COMP=ILCS in the KDCDEF generation. It is not necessary to issue PROGRAM statements for the various programming languages.

B

B

B

Prerequisites for transaction codes

Please note the following for the dynamic configuration of transaction codes:

- Transaction codes for program units that use an X/Open program interface can only be created dynamically if at least one transaction code has been configured for an X/Open program unit in the KDCDEF generation (TAC statement with API≠KDCS).
- If you want to divide the transaction codes into TAC classes to control job processing, then you must create at least one TAC class in the KDCDEF generation. TAC classes can be created in two ways in the KDCDEF generation:
 - You generate a transaction code, and you specify a TAC class in the TACCLASS operand (TAC statement) that is then implicitly created by KDCDEF.
 - If you run the application **without** priority control (the application does not contain any TAC-PRIORITIES statements), then you can create the TAC classes by writing a TACCLASS statement.

If you created a TAC class in the KDCDEF generation, then the transaction codes you enter dynamically can be assigned to any TAC class between 1 and 16. The TAC classes are then created implicitly by openUTM. These TAC classes can be administered.

If the application is generated **without** TAC-PRIORITIES, then openUTM assigns the process numbers (TASKS) for implicitly created TAC classes as follows:
1 for dialog TAC classes (classes 1 through 8)
and 0 for asynchronous TAC classes (classes 9 through 16).

Asynchronous TAC classes (9 through 16) are only created by openUTM, however, if you have set ASYNTASKS > 0 in the MAX statement in the generation.

- In applications with TAC classes **without** priority control, you can only dynamically create transaction codes that start the program unit runs with blocking calls when TAC classes have been statically generated with PGWT=YES (dialog and/or asynchronous TAC classes).
Dialog and asynchronous TAC classes with PGWT=YES must therefore be generated explicitly in the KDCDEF generation with TACCLASS statements.
You must also set MAX TASKS-IN-PGWT > 0.
- In applications **with** priority control (with TAC-PRIORITIES statements) , you can only dynamically create transaction codes that start the program unit runs with blocking calls (TAC ...,PGWT=YES) when MAX TASKS-IN-PGWT>0 was set in the KDCDEF generation.

Prerequisites for user IDs

User IDs can only be entered dynamically if your application is generated with user IDs. In this case, your KDCDEF generation must contain at least one USER statement. At least one user ID must be configured with administration authorization, so that the calls for dynamic administration can be executed under this user ID.

B If user IDs with ID cards are also to be configurable, the percentage of table locations that
B can be occupied by user IDs with ID card must be explicitly specified when reserving the
B table locations with the RESERVE statement.

B You must generate the length of the ID card information statically in the KDCDEF generation
B using the MAX statement for user IDs with ID cards:
B `MAX...,CARDLTH=length`

8 The tool KDCUPD – updating the KDCFILE

You can use the KDCUPD tool after regenerating your UTM application to transfer important user data and administration information of the production application from the old KDCFILE to the new one. In addition, you can use the KDCUPD update tool to switch from an older openUTM version to the current new openUTM version without losing the data from the previous production application in the KDCFILE.

The same applies to UTM cluster applications, both for the KDCFILES of the node applications and for the user data and management information in the UTM cluster files created during generation.

You can use the KDCUPD statement TRANSFER to control which data are to be transferred. KDCUPD automatically carries out a consistency check for the KDCFILE files prior to transfer.

You can use the KDCUPD statement CHECK to check the completeness and consistency of the KDCFILE files of an application without transferring data.



In the case of UTM cluster applications, a distinction is made in the KDCUPD run depending on whether the data of a KDCFILE or the data from the UTM cluster files is to be transferred. For details, see the [section “Updating the KDCFILE and UTM cluster files for UTM cluster applications” on page 590](#).

8.1 Overview

This section provides an overview of

- Version upgrades
- Prerequisites
- Data backups
- Scope of transfer, i.e. what data is transferred

8.1.1 Supported upgrades

You can use the KDCUPD utility program to transfer data from applications from openUTM versions 5.3, 6.0 or 6.1. The KDCUPD utility program from openUTM V6.1 also supports the following upgrades:

openUTM V5.3 → openUTM V6.1 (standalone applications only)
openUTM V6.0 → openUTM V6.1
openUTM V6.1 → openUTM V6.1

When changing versions, you must create a new KDCFILE with the KDCDEF of the new openUTM version.

A changeover from a standalone UTM application to a UTM cluster application or vice versa is only possible within Version 6.1.

KDCUPD does not support a change from a newer openUTM to an older one.

8.1.2 Prerequisite for using KDCUPD

The prerequisites for running KDCUPD are:

- You have used the KDCDEF to create a new KDCFILE.

If the application is a UTM cluster application and if a cluster update is to be performed then it is necessary to use the KDCDEF generation tool to create new UTM cluster files as well as the new KDCFILE.
- The application was terminated normally (e.g. with the administration command KDCSHUT N, W or G). If the application was terminated abnormally, then you must execute a warm start beforehand and then terminate the application normally. In the case of a cluster update, all the node applications must have been terminated normally.

In the case of a cluster update, all the node applications must have been terminated normally.

8.1.3 Backing up data

Before you start your work, please read the following notes:



CAUTION!

- When you create the new KDCFILE with KDCDEF, you must ensure that you **do not accidentally overwrite the old KDCFILE** and thus destroy important application data!
- The records in user log files (USLOG files) must be saved before the application is restarted because openUTM overwrites the current USLOG file generation from the beginning after a KDCUPD run.

There are a number of ways of avoiding overwriting the KDCFILE:

- As shown here, you create the new KDCFILE before terminating the application. You use the same base name, but set up the KDCFILE under another ID (BS2000/OSD) or in another directory (Unix systems, Windows systems). After you terminate the application, you must rename or copy the files for the subsequent KDCUPD run.
- First terminate the application and then rename the old KDCFILE and all associated files by changing the base name. Alternatively, copy the old KDCFILE and all associated files to a different ID (BS2000/OSD) or to a different directory (Unix systems, Windows systems). Then start the KDCDEF run to generate the new KDCFILE with the same base name.

Specify the following in the subsequent KDCUPD run:

- KDCFILE OLD= *base_name-renamed/copied-KDCFILE*
- KDCFILE NEW= *base_name-new-KDCFILE*

- Use a new base name for the new KDCFILE and work with this name in the KDCUPD run. When you subsequently start the application, you can either use the new base name or continue to use the previous base name after copying and renaming the files.

8.1.4 What data does KDCUPD transfer?

This section lists the data that is transferred, indicates the dependencies of the UTM variants and generation parameters and describes in greater detail which user data is always transferred and which it might sometimes not be possible to transfer.

Transfer in standalone applications

The data that KDCUPD transfers from the old KDCFILE to the new one depends on the variant of the UTM application, see also [section “Transfer of user data” on page 583](#):

- UTM-S and UTM-F applications

KDCUPD only transfers certain changes to the administration data:

- passwords and RSA keys
- version number of load modules and Locales under BS2000/OSD
- version number of shared objects under Unix systems
- version number of DLLs under Windows systems.

B
X
W

In addition, all available RSA keys of levels 1 to 4 are also transferred in a KDCUPD run. Active keys and keys created using administration facilities but not yet activated are transferred. If, in the old KDCFILE, there are no RSA keys in an encryption level, then nothing is transferred for this level. It can therefore happen that RSA keys generated for this encryption level in the new KDCFILE are not overwritten with 0.

- UTM-S applications

KDCUPD also transfers administration data and current user data such as global secondary storage areas, asynchronous messages, TLS or ULS areas, and service-specific information etc. from the previous KDCFILE to a newly generated KDCFILE. In the data transfer, the KDCUPD checks whether the owner, the destination or the initiator of the data is missing in the new KDCFILE or if it was deleted by the administration in the previous application run. In this case, KDCUPD does not transfer the data and logs this event.

Transfer in UTM cluster applications

In UTM cluster applications, the scope of the transferred data also depends on whether you are performing a node update or a cluster update.

Cluster update

When a cluster update is performed in a UTM cluster application, the management and user data for the GSSB, ULS and the service-specific information from the previous UTM cluster files is imported into the new UTM cluster files irrespective of the variant of the UTM application. If data cannot be transferred, for example because the owner of the service-specific data is not present in the new UTM cluster files, then this is logged.

Node update

In the case of a node update, the data that KDCUPD transfers from the old to the new KDCFILE depends on the variant of the UTM application:

- UTM-S and UTM-F applications

KDCUPD only transfers certain changes to the administration data:

- passwords and RSA keys
- version number of load modules and Locales under BS2000/OSD
- version number of shared objects under Unix systems
- version number of DLLs under Windows systems.

B
X
W

All available RSA keys of levels 1 to 4 are also transferred in a KDCUPD run. Active keys and keys created using administration facilities but not yet activated are transferred. If, in the old KDCFILE, there are no RSA keys in an encryption level, then nothing is transferred for this level. It can therefore happen that RSA keys generated for this encryption level in the new KDCFILE are not overwritten with 0.

- UTM-S applications

KDCUPD transfers administration data and current user data such as asynchronous messages, TLS areas from the previous KDCFILE to a newly generated KDCFILE. In the data transfer, the KDCUPD checks whether the owner, the destination or the initiator of the data is missing in the new KDCFILE or if it was deleted by the administration in the previous application run. In this case, KDCUPD does not transfer the data and logs this event.

For further details, see [section “Updating the KDCFILE and UTM cluster files for UTM cluster applications” on page 590](#). The effect of the individual parameters in node updates and cluster updates can be found in the description of the TRANSFER statement, see [section “TRANSFER - control the data transfer of the user data” on page 606 ff.](#)

8.1.4.1 Changing generation parameters

KDCUPD compares the generations of the two KDCFILEs. Depending on the results of these checks, KDCUPD cannot transfer some items of data and in some cases must reject transfer completely.

If the KDCUPD run detects that the database configurations of the two generations are incompatible then an error message (K307) is output.

B *No transfer*

B When generating new KDCFILE the user is basically permitted to change all the generation
B parameters compared with the old KDCFILE. However, there are some exceptions which
B apply only to UTM-S in the case of standalone applications and to UTM-S and UTM-F in
B the case of UTM cluster applications.

B The entire upgrade is denied by KDCUPD for the UTM-S variant if there are the following
B differences between certain generation parameters in the new and old generation because
B running an application with the KDCFILE would lead to errors.

B ● Old KDCFILE generated with formatting, new KDCFILE without.

B ● Different database systems or different number of database systems, if they were
B defined with DATABASE statements.

B The upgrade is not denied in the following cases:

B – No database system was generated in the old KDCFILE one or two database
B systems are generated in the new KDCFILE.

B – In the old KDCFILE, the Oracle database system was generated with
B DATABASE, TYPE=ORACLE, and in the new KDCFILE with DATABASE,
B TYPE=XA.

B ● The same database systems are entered in the generations, but the DATABASE state-
B ments are not in the same sequence.

B In the case of standalone applications of the UTM-F variant, differences of this type do not
B prevent KDCUPD from carrying out the transfer.

Limited transfer

There are generation differences between the old and new KDCFILE which in principle permit transfer, but for which individual messages or data areas cannot be transferred. KDCUPD logs events such as these to SYSOUT or SYSLST (on BS2000/OSD) or to stdout or stderr (on Unix systems and Windows systems) and continues transfer to the new KDCFILE.

Examples

If an LTERM partner is no longer defined in the new KDCFILE, KDCUPD cannot transfer any FPUT messages and TLS areas for these LTERM partners.

If the communication area of a dialog service is larger than the maximum communication area length (operand MAX KB=...) in the new KDCFILE, KDCUPD rejects transfer of this service.

In general, the generation values of the new KDCFILE apply when transferring with KDCUPD.

8.1.4.2 Transfer of user data

The transfer of user data can be controlled using the KDCUPD TRANSFER statement.

Please note that in the case of standalone applications as well as of node updates in UTM cluster applications, the following remarks only apply to UTM-S.

User data always transferred by KDCUPD

KDCUPD always transfers the following data of a KDCFILE, irrespective of the specifications in the TRANSFER control statement:

- asynchronous messages in USER queues when the user, generating LTERM, and generating USER also exist in the new KDCFILE

User data which KDCUPD transfers optionally

The TRANSFER control statement allows you to control which of the following data KDCUPD is to transfer to the new KDCFILE or the UTM cluster files:

- Dialog services started by a terminal or a TS application of type APPLI.
- Dialog services started by a UPIC client.
- Dialog services started by a TS application of type SOCKET.
- Passwords for the user IDs and - if generated - the validity period remaining, minimum wait time until the next password change, and the password history
- Secondary storage area GSSB, TLS, and ULS
- Queued output jobs
- Queued messages to local asynchronous services and TAC queues as well as open asynchronous services
- Queued messages to local partners
- Queued messages to remote partners

- Queued messages to temporary queues
- Current version number of the loadable objects (load objects under BS2000/OSD, shared objects on Unix systems, DLLs on Windows systems)
- B** ● The locales of user IDs (only on BS2000/OSD)

When services are transferred, all service-specific data is transferred:

- Local secondary storage area data
- Saved dialog messages
- Communication areas
- Batch stacked services (only for standalone applications)

Data not transferred by KDCUPD

The following is not transferred by KDCUPD:

- Objects which have been added with the administration function such as new USERS.
- Data belonging to open, distributed services.
KDCUPD does not issue a message that this data was not transferred!
- Open dialog services of users when the user does not exist in the new KDCFILE.
- Open asynchronous services when the user who started the service or the LTERM or (OSI-)LPAP partner from which the asynchronous service was started does not exist in the new KDCFILE.
- Queued messages when the destination of the message, the user who generated the message, or the LTERM or (OSI-)LPAP partner from which the asynchronous service was started does not exist in the new KDCFILE.
- The TLS or ULS storage areas when the corresponding LTERM, (OSI-)LPAP, the associated USER, or the associated session or association does not exist in the new KDCFILE.
- Service stacks in UTM cluster applications.

8.2 Updating the KDCFILE for standalone UTM applications

Steps



Before performing the operations below, please read section “[Backing up data](#)” on [page 579](#)!

Steps 1 to 5 on the following pages explain how to use the tool KDCUPD to update the KDCFILE.

1. Create the new KDCFILE with KDCDEF
2. Terminate the application normally
3. Rename/copy the old KDCFILE and back up the user log file
4. Call KDCUPD
5. Start application

These steps are described in detail on the following pages. The method shown here is only one of several possibilities. In all cases, you must ensure that the KDCFILE is not overwritten, see warning on [page 579](#).

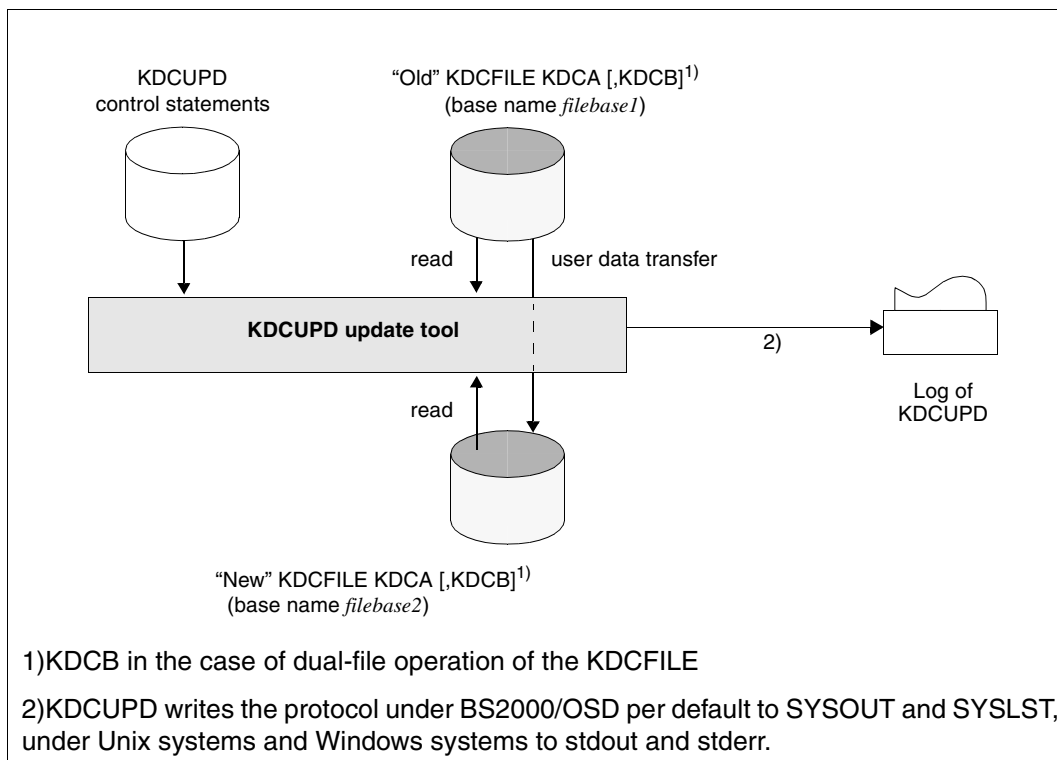


Figure 20: Updating the KDCFILE

1. Create the new KDCFILE with KDCDEF

You can change the configuration of your application in the KDCDEF run, i.e. you can define new partner applications and connections, for example, delete existing ones or change application properties, etc.

It is possible to switch from single-file to dual-file operation. In addition, the “new” KDCFILE can be divided into several files, or the number of files can be changed. When dual-file operation is used and/or the new KDCFILE is distributed across several files, all these files must exist and KDCUPD must be able to access them.

2. Terminate the application normally

Terminate the application normally, see [section “Prerequisite for using KDCUPD” on page 578](#).

The application must be terminated normally before calling KDCUPD so that the old KDCFILE is in a consistent state.

Make sure that the user log file exists and is not locked when the application is terminated. openUTM stores the user log records (LPUT calls) temporarily and does not write them to the file immediately. When the application is terminated normally, an attempt is made to write these records to the current user log file. If this attempt fails, the records remain in the old KDCFILE.

KDCUPD does **not** transfer these LPUT records to the new KDCFILE. But KDCUPD indicates with a warning message (K314) that the data get lost.

3. Copy the KDCFILE and back up the user log file

After terminating the application, you must copy or rename the current KDCFILE, for instance to OLD.KDCA. Then you assign the ‘correct’ name to the new KDCFILE.

When dual-file operation is used and/or the current KDCFILE is distributed across several files, all these files must be copied or renamed using the same filebase name, and KDCUPD must be able to access them.

Back up any existing user log file, as this file is overwritten when the system is restarted after a KDCDEF or KDCUPD run.

4. Call KDCUPD

In a subsequent KDCUPD run, specify the base name of the copied KDCFILE (including the user ID under BS2000) in the control statement KDCFILE OLD= and specify the base name of the newly generated KDCFILE in the control statement KDCFILE NEW=.

B *BS2000/OSD:*

B Before calling KDCUPD you must do the following:

B ● The library SYSLNK.UTM.061 must be set as the tasklib:

B /SET-TASKLIB LIBRARY=\$userid.SYSLNK.UTM.061

B The library of the UTM version with which the new KDCFILE was generated must be assigned. If no library or an incorrect library was assigned, the DLL outputs a corresponding message.

B ● Process switch 3 should be set to OFF:

B /MODIFY-JOB-SWITCHES OFF=3

B KDCUPD is called with /START-EXECUTABLE-PROGRAM. KDCUPD reads control statements from SYSDTA. A description of the KDCUPD control statements can be found as of [page 600](#).

B KDCUPD is called as follows:

```

B START-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM
B           (LIBRARY=$userid.SYSLNK.UTM.061.UTIL, ELEMENT-OR-SYMBOL=KDCUPD)
B :
B :       KDCUPD control statements
B :
B *END

```

B KDCUPD outputs the procedure for logging purposes to SYSLST and/or SYSOUT (see control statement LIST on [page 604](#)).

X/W *Unix systems, Windows systems:*

X/W The subdirectory DUMP must exist in the base directory of the old KDCFILE (*filebase1*) as well as in the base directory of the new KDCFILE (*filebase2*) before KDCUPD is called.

X/W DUMP is used for diagnostic purposes.

X/W You start KDCUPD under Unix systems from the shell. You must call KDCUPD in a DOS window in Windows systems. Enter the following command to do this:

X Unix systems: *utmpath/ex/kdcupd 1>upd.out*

W Windows systems: *utmpath\ex\kdcupd 1>upd.out.txt*

X/W Note the following:

- X/W ● You should **always** redirect the output to *stdout* to a file (`upd.out` or `upd.out.txt` in the examples above).
- X/W ● You may **not** redirect *stderr* (command mode) because KDCUPD asks you to enter the control statements via *stderr* (e.g. the `*'` of the KDCUPD input mode is output to *stderr*).

X/W KDCUPD reads the control statements from *stdin*. A description of the control statements for KDCUPD can be found in [section “Control statements for KDCUPD” on page 600](#).

X/W KDCUPD outputs the procedure for logging purposes to *stdout* and/or *stderr* (see control statement LIST on [page 604](#)).

5. Start the application

The base name (*filebase2*) of the new KDCFILE must be specified in the start parameter `FILEBASE=`. Modify your start procedure if you have assigned a new base name to the new KDCFILE when creating it.

If the data has been transferred to the new KDCFILE with KDCUPD and you restart the application, then every user can continue to work as if the application was terminated normally and then restarted.

8.3 Updating the KDCFILE and UTM cluster files for UTM cluster applications

KDCUPD allows you to update and convert UTM cluster applications when performing the following actions:

- Online update of a UTM cluster application V6.1
- Update generation of a UTM cluster application V6.1, see [page 592](#)
- Conversion of a UTM cluster application V6.1, see, siehe [page 594](#)

When performing a KDCUPD run, you can choose between a node update and and a cluster update:

- In the case of a **node update**, you update the KDCFILE of a node application.
- In the case of a **cluster update**, you update the UTM cluster files created during generation.

You can control these two variants using the KDCFILE and CLUSTER-FILEBASE statements.



Before performing a KDCDEF or KDCUPD run, please read section “[Backing up data](#)” on [page 579](#)!

The following sections describe the KDCUPD statements that are required for these functions and provide information on how you must first use KDCDEF to create the generation files. In addition, further activities are required depending on the situation, e.g. starting or terminating node applications or the UTM cluster application or adapting the start parameters.



For details see the relevant openUTM manual “Using openUTM Applications”, section “Update generation in the cluster”.

8.3.1 Online update of a UTM cluster application

When you perform an online update, only the KDCFILE is regenerated and no changes are made to the UTM cluster files. As a result, the changes to the generation apply only to the KDCFILE.

You can perform an online update while the cluster application is running, i.e. at least one node application is always active during the update. To perform an online update, you can terminate each node application in succession, run a node update to transfer the node-specific data and then restart the node applications. The node-specific data includes, for example, TLS areas, asynchronous messages to LTERM, OSI(-LPAP), asynchronous TACs and TAC, USER and temporary queues or open asynchronous services.

Please note that global administration of all applications of a cluster and an online inverse KDCDEF run are not possible until all active node applications have been updated to the same generation status. Local administration of individual node applications, however, can be carried out at any time.



CAUTION!

After a node application has been restarted on the basis of a newly generated KDCFILE, it is not possible to start other node applications using a KDCFILE from an older generation run!

Proceed as follows to perform an online update:

KDCDEF run

Use KDCDEF to generate a new initial KDCFILE. When you do this, you must specify the cluster filebase name of the current UTM cluster files. You must **not** specify OPTION GEN=CLUSTER.

KDCUPD runs

1. First node update

Make the old KDCFILE and the new KDCFILE file under the base name specified below.

Perform the KDCUPD run with the following statements:

```
KDCFILE OLD=filebase-old,NEW=filebase-new
TRANSFER ...
```

Explanation

filebase-old

Base name of the node application's old KDCFILE.

filebase-new

Base name of the new KDCFILE generated using KDCDEF and copied for the node application. KDCUPD transfers the data from the old KDCFILE to the node application's new KDCFILE. You use the TRANSFER statement to specify the scope of the data to be transferred.

2. Subsequent node updates

Perform the actions described in step 1 for all the other node applications without delay.

8.3.2 Update generation for a UTM cluster application

When you perform an update generation for a UTM cluster application, the UTM cluster files (cluster configuration file, cluster user file, cluster page pool etc.) are also regenerated. To do this, the UTM cluster application must first have been completely terminated.

Before a cluster update can be performed, all the node applications must have been terminated normally.

A new variant of KDCUPD can now be used during update generations. This is the so-called cluster update. When you perform a cluster update you require the old UTM cluster files and the new UTM cluster files generated by KDCDEF in addition to the old and new KDCFILE.

To perform a cluster update, you must specify not only the KDCUPD statement KDCFILE but also the statement CLUSTER-FILEBASE. You can specify the initial KDCFILE or any other node KDCFILE in the KDCFILE statement.

During a cluster update, data that applies globally in the cluster such as GSSB and ULS areas is transferred together with service-specific information and passwords. The data that applies globally throughout the cluster is also transferred in the case of UTM-F. You can also use the TRANSFER statement to control the data that is transferred during a cluster update.

Proceed as follows to perform an update generation:

KDCDEF run

Use KDCDEF to generate the new initial KDCFILE for the UTM cluster application, including the UTM cluster files. To do this, specify GEN=(CLUSTER,KDCFILE) in the OPTION statement.

KDCUPD runs

1. Cluster update

Make the old and new UTM cluster files and the old and new KDCFILE available under the base names specified below. For the old and new KDCFILE, you can use either the initial KDCFILE or the KDCFILE of a node application. In this case, the KDCFILES are used only for the program run and for a variety of checks. The content of the old KDCFILE is **not** transferred and the new KDCFILE is not changed.

Perform the KDCUPD run with the following statements:

```
CLUSTER-FILEBASE OLD=cluster-filebase-old,NEW=cluster-filebase-new  
KDCFILE OLD=filebase-old,NEW=filebase-new  
TRANSFER ...
```

Explanation

cluster-filebase-old

Base name of the old UTM cluster files.

cluster-filebase-new

Base name of the new UTM cluster files generated by KDCDEF. KDCUPD transfers the data that is valid globally in the cluster from the old UTM cluster files to the new UTM cluster files. You use the TRANSFER statement to specify the scope of the data to be transferred.

filebase-old

Base name of the old KDCFILE in the UTM cluster application.

filebase-new

Base name of the new KDCFILE in the UTM cluster application.

2. First node update

Perform the node update in the same way as an online update, see [page 591](#).

After the KDCUPD run, you can start the node application.

3. Subsequent node updates

Repeat step [2](#) for all the other node applications.

8.3.3 Converting a UTM cluster application

KDCUPD allows you to convert UTM cluster applications when performing the following actions:

- Conversion from a standalone application V6.1 to a UTM cluster application V6.1
- Conversion from a UTM cluster application V6.0 to V6.1, see [page 596](#)
- Conversion from a UTM cluster application V6.1 to a standalone UTM application V6.1, see [page 597](#)

8.3.3.1 Conversion from a standalone UTM application to a UTM cluster application

Standalone UTM applications can only be converted directly to UTM cluster applications in the case of UTM applications as of version 6.1.

If you want to convert a standalone UTM application V5.3 or V6.0 into a UTM cluster application then you must first convert it into a version 6.1 standalone application. For details, see [section “Updating the KDCFILE for standalone UTM applications” on page 585](#).

KDCDEF run

Use KDCDEF to generate the initial KDCFILE for the UTM cluster application, including the UTM cluster files. To do this, specify GEN=(CLUSTER,KDCFILE) in the OPTION statement.

KDCUPD runs

1. Cluster update

Make the new UTM cluster files and the old and new KDCFILE available under the base names specified below. For the new KDCFILE, you can use either the initial KDCFILE or the KDCFILE of a node application. In this case, the KDCFILES are only used for various checks. The content of the new KDCFILE is not changed.

Perform the KDCUPD run with the following statements:

```
CLUSTER-FILEBASE NEW=cluster-filebase  
KDCFILE OLD=filebase-old,NEW=filebase-new  
TRANSFER ...
```

Explanation

cluster-filebase

Base name of the new UTM cluster files generated by KDCDEF. KDCUPD transfers the data that applies globally at cluster level from the old KDCFILE of the standalone application to the UTM cluster files. You use the TRANSFER statement to specify the scope of the data to be transferred.

filebase-old

Base name of the old KDCFILE in the standalone application.

filebase-new

Base name of the new KDCFILE in the UTM cluster application.

2. Node update

Perform a KDCUPD run with the following statements for a node application's KDCFILE:

```
KDCFILE OLD=filebase-old,NEW=filebase-new  
TRANSFER ...
```

Explanation

filebase-old

Base name of the old KDCFILE in the standalone application.

filebase-new

Base name of the node application's KDCFILE. KDCUPD transfers the data from the old KDCFILE to the node application's KDCFILE. You use the TRANSFER statement to specify the scope of the data to be transferred.

**CAUTION!**

You can only perform a KDCUPD run for a node application!

8.3.3.2 Converting a UTM cluster application from V6.0 to V6.1

When converting a UTM cluster application from V6.0 to V6.1, the only global cluster-level data that you can transfer are the passwords and locales. GSSB, ULS and service data is not transferred even if the old UTM cluster application was generated with GLOBAL-UTM-DATA=YES and/or USER-RESTART=YES.

KDCDEF run

Use KDCDEF to generate the initial KDCFILE for the UTM cluster application, including the UTM cluster files. To do this, specify GEN=(CLUSTER,KDCFILE) in the OPTION statement.

KDCUPD runs

1. Cluster update

Before the last node application is terminated, the administration facilities must be used to read the current values for all users, e.g. using WinAdmin. This transfers the user passwords from the cluster user file into the KDCFILE of this node application. Use this node application's KDCFILE ("old" KDCFILE) for the cluster update. For the new KDCFILE, you can use either the initial KDCFILE or the KDCFILE of a node application.

Perform the KDCUPD run with the following statements:

```
CLUSTER-FILEBASE NEW=cluster-filebase  
KDCFILE OLD=filebase-old,NEW=filebase-new  
TRANSFER ...
```

Explanation

cluster-filebase

Base name of the new UTM cluster files generated by KDCDEF. KDCUPD transfers the locales (BS2000/OSD) and passwords from the node application's old KDCFILE of Version 6.0 to the new UTM cluster files unless you have specified otherwise in the TRANSFER statement.

filebase-old

Base name of the selected node application's old KDCFILE.

filebase-new

Base name of the new KDCFILE in the UTM cluster application. This KDCFILE is required only for checking purposes and no data is taken over into the new KDCFILE.

2. Node update

Perform the KDCUPD run with the following statements for all node applications:

```
KDCFILE OLD=filebase-old,NEW=filebase-new  
TRANSFER ...
```

Explanation

filebase-old

Name of the old KDCFILE of the selected node application.

filebase-new

Name of the new KDCFILE generated using KDCDEF and copied for the node application. KDCUPD transfers the data from the node application's old KDCFILE to the new KDCFILE. You use the TRANSFER statement to specify the scope of the data to be transferred.

8.3.3.3 Converting a UTM cluster application to a standalone UTM application

If you want to convert a UTM cluster application of V6.1 into a standalone V6.1 application then you can perform either a cluster update or a node update, but not both. This is due to the fact that KDCUPD is only able to transfer data to a newly generated KDCFILE.

When you perform a cluster update, you can only transfer data that applies globally to the cluster such as passwords, locales, GSSB, ULS and service-specific data. In the case of a node update, you can only transfer local, node-level data such as TLS, asynchronous messages etc.

KDCDEF run

Use KDCDEF to generate the KDCFILE for the standalone application, i.e. GEN=CLUSTER must not be specified in the OPTION statement.

KDCUPD run

You can only perform one KDCUPD run. If you perform a cluster update then you cannot perform a node update and vice versa.

- Cluster update

Make the old UTM cluster files and the old and new KDCFILE available under the base names specified below. For the old KDCFILE, you can use either the initial KDCFILE or the KDCFILE of a node application. In this case, the KDCFILES are only used for various checks. The content of the old KDCFILE is **not** transferred.

Perform the KDCUPD run with the following statements:

```
CLUSTER=FILEBASE OLD=cluster-filebase-old
KDCFILE OLD=filebase-old,NEW=filebase-new
TRANSFER ...
```

Explanation

cluster-filebase-old

Base name of the old UTM cluster files. KDCUPD transfers the data that applies globally at cluster level to the KDCFILE of the new standalone application. You use the TRANSFER statement to specify the scope of the data to be transferred.

filebase-old

Base name of the selected KDCFILE in the UTM cluster application (initial KDCFILE or KDCFILE of a node application).

filebase-new

Name of the KDCFILE of the standalone application. KDCUPD transfers the data of the UTM cluster files to the new KDCFILE.

- Node update



To transfer as much node-specific data as possible during the node update, it may be advisable, in the last running node application, to initiate an online import for all the other node applications and then to use the KDCFILE of this last node application for the KDCUPD run.

Perform the KDCUPD run with the following statements:

```
KDCFILE OLD=filebase-old,NEW=filebase-new
TRANSFER ...
```

Explanation

filebase-old

Name of the KDCFILE of the selected node application.

filebase-new

Name of the KDCFILE that was generated with KDCDEF. KDCUPD transfers the data from the KDCFILE of the node application to the KDCFILE of the standalone application. You use the TRANSFER statement to specify the scope of the data to be transferred.

8.4 Control statements for KDCUPD

KDCUPD knows the following control statements for data transfer:

Statement	Meaning
TRANSFER	Control the transfer of the data from the old KDCFILE to the new KDCFILE
KDCFILE	Specify the base name of the newly generated and the old KDCFILE
CLUSTER-FILEBASE	Specify base name of old and new UTM cluster files
CATID	The catid of the old and new KDCFILE are specified under BS2000/OSD
LIST	Control the runtime log
END	Terminate input and start processing

B

KDCUPD knows the following control statements for the consistency check:

Statement	Meaning
CHECK	Check the consistency of the KDFILE of a UTM application
LIST	Control the runtime log
END	Terminate input and start processing

The control statements are read from SYSDTA under BS2000/OSD or from *stdin* (command prompt) under Unix systems and Windows systems.

The control statements KDCFILE, CATID, LIST and CHECK may only be entered once per KDCUPD run. The input must be contained in a single line. Multiple (separate single) lines can be entered for the control statement TRANSFER.

Example

1. KDCUPD is to transfer the data from the old KDCFILE (base name B00K01) of a standalone UTM-S application to the new KDCFILE (base name B00K02). All data is to be transferred except for the asynchronous messages intended for the communication partners (LPAP and LTERM) that are still located in the message queues of the old KDCFILE. KDCUPD is to output the successful transfer messages only to SYSLST (BS2000/OSD) or *stdout* (Unix systems, Windows systems).

Input under BS2000/OSD	Input under Unix systems, Windows
*KDCFILE NEW=BUCH02,OLD=BUCH01 *CATID OLD=(20SN,20PN),NEW=(20SN,20PN) *TRANSFER ASYNLPAP=NO *TRANSFER ASYNTERM=NO *LIST PROTOCOL=SYSLST *END	*KDCFILE NEW=BUCH02,OLD=BUCH01 *TRANSFER ASYNLPAP=NO *TRANSFER ASYNTERM=NO *LIST PROTOCOL=STDOUT *END

2. If you only enter the mandatory parameters in all statements, then KDCUPD attempts to transfer everything and outputs the log to SYSOUT and SYSLST or to *stdout* and *stderr*. You must specify the following to transfer all data:

under BS2000/OSD	under Unix systems, Windows
*KDCFILE NEW=BUCH02,OLD=BUCH01 *CATID OLD=(20SN,20PN),NEW=(20SN,20PN) *END	*KDCFILE NEW=BUCH02,OLD=BUCH01 *END

CATID - define Catid of the old and the new KDCFILE

B The CATID statement specifies the catalog ID of the old and new KDCFILE. You must
B specify at least one of the operands OLD or NEW.

B CATID_ [OLD=(catalog_A,catalog_B)]

B [, NEW=(catalog_A,catalog_B)]

B OLD=(catalog_A,catalog_B)

B Catids for the old (previously used) KDCFILE

B NEW=(catalog_A,catalog_B)

B Catids for the new KDCFILE

B If you specify only *catalog_A*, this Catid is assigned to all parts of the
B KDCFILE. If you are working with Catids, the base name (*filebase1/2*) must
B be specified in the KDCFILE statement without a catalog ID.

CHECK - check the consistency of the KDCFILE

The CHECK statement causes KDCUPD to check the KDCFILE file(s) of an application for consistency (completeness, identical generation and processing status). No data is transferred.

CHECK_ filebase

filebase Base name of the KDCFILE
(KDCDEF control statement MAX KDCFILE=*filebase*)

B *BS2000/OSD*:

B With dual-file operation, *filebase* must be specified with the catalog ID of the
B A files.

CLUSTER-FILEBASE - Specify the base names of the old and new UTM cluster files

You use the CLUSTER-FILEBASE statement to inform KDCUPD of the base names of the old and new UTM cluster files.

B CLUSTER-FILEBASE= [NEW=cluster_filebase2]

B [, OLD=cluster_filebase1]

NEW=cluster_filebase2

Base name of the newly generated UTM cluster files.

You must not specify this parameter when converting a UTM cluster application to a standalone application.

OLD=cluster_filebase1

Base name of the previously used UTM cluster files.

You must not specify this parameter when converting a standalone application to a UTM cluster application.



The old and new UTM cluster files must have different base names. You can do this in two ways:

- For the new KDCDEF generation, enter a base name that is different from that in the old KDCDEF generation (KDCDEF statement CLUSTER; operand CLUSTER-FILEBASE).
- In the KDCDEF generation, leave the base name in CLUSTER CLUSTER-FILEBASE= unchanged. When you do this, rename the previously used UTM cluster files before the KDCUPD run so that they have a different base name.

END - terminate input and start processing

The END statement terminates the entry of parameters and starts processing. The KDCUPD control statements must be terminated with END.

END

KDCFILE - specify the base name of the old and new KDCFILE

The KDCFILE statement passes the base name of the old and new KDCFILE to KDCUPD.

```
KDCFILE_      NEW=filebase2  
              , OLD=filebase1
```

NEW=filebase2

Base name of the newly generated KDCFILE

B
B
B
B

BS2000/OSD:

In the case of dual-file operation, the catalog IDS for the A and B files can be assigned with the CATID statement. In this event, you must specify the base name without the catalog ID.

OLD=filebase1

Base name of the old KDCFILE

LIST - control the runtime log

The LIST statement controls output of the positive and negative transfer messages as well as messages K305 and K306 for the number of pages used in the page pool.

```
LIST_ [ ERRORS={ BOTH | SYSOUT1 | SYSLST1 | STDERR2 | STDOUT2 } ]
      [ ,INFO={ SHORT | LONG | NO | } ]
      [ ,PROTOCOL={ BOTH | NO | SYSOUT1 | SYSLST1 | STDERR2 | STDOUT2 } ]
```

B ¹ SYSLST and SYSOUT are only permitted under BS2000/OSD.

X/W ² STDERR and STDOUT are only permitted under Unix systems and Windows

ERRORS= Controls the output of negative transfer messages

BOTH Data is logged to SYSOUT and SYSLST (under BS2000/OSD) or to *stderr* and *stdout* under Unix systems and Windows.

Default: BOTH

B **SYSOUT** Data is logged to SYSOUT.

B **SYSLST** Data is logged to SYSLST.

X/W **STDOUT** Data is logged to *stdout*.

X/W **STDERR** Data is logged to *stderr*.

INFO= Controls the output of messages K305 and K306 for the number of pages used in the page pool of the new KDCFILE

SHORT At the end of the log for the successful transfer messages, K306 messages are used to output an overview of the number of pages used in the page pool. An overall view and a view with the number of pages used by each of the GSSB, QUEUE, LTERM, LPAP, TAC, USER, and ASYNVG object types (if they use pages in the page pool) are output.

In the page pool usage view the following are displayed for each type of object:

GSSB	Data of the GSSB storage area
QUEUE	Asynchronous messages of the temporary queue and the management information required for administration
LTERM	TLS blocks and asynchronous messages (including management information) of the LTERM
LPAP	TLS blocks and asynchronous messages (including management information) of the LPAP or OSI-LPAP
TAC	Asynchronous messages (including management information) of the TAC
USER	ULS blocks, asynchronous messages (including management information) and dialog service information of the user, ULS blocks of the LSES or OSI-ASS.
ASYNVG	Service data (including any LSSBs that are present) of all the user's open asynchronous services

Standard: SHORT

LONG	In a K305 message, the page pool usage is also output for each object of this type after the last positive transfer message as long as at least one page pool page is used.
NO	Messages K305 and K306 are deactivated.

PROTOCOL= Controls the output of positive transfer messages and the messages for the page pool usage (see INFO).

BOTH	Data is logged to SYSOUT and SYSLST (under BS2000/OSD) or to <i>stderr</i> and <i>stdout</i> under Unix systems and Windows systems.
------	--

Default: BOTH

NO	No positive transfer messages are output and no messages for the page pool usage.
----	---

B	SYSOUT	Data is logged to SYSOUT.
B	SYSLST	Data is logged to SYSLST.
X/W	STDOUT	Data is logged to <i>stdout</i> .
X/W	STDERR	Data is logged to <i>stderr</i> .

TRANSFER - control the data transfer of the user data

The TRANSFER specifies the user data KDCUPD is to transfer to the new KDCFILE.

```

TRANSFER_  [ ASYNLPAP=YES | NO ]
            [ ,ASYNTACS=YES | NO ]
            [ ,ASYNTERM=YES | NO ]
            [ ,DIALOGS=YES | NO ]
            [ ,LOCALE=YES | NO ]
            [ ,PASS=YES | NO ]
            [ ,PROG-VER=YES | NO ]
            [ ,SOCKET-DIALOGS=YES | NO ]
            [ ,STORAGES=YES | NO ]
            [ ,UPIC-DIALOGS=YES | NO ]

```

B

If you omit the TRANSFER statement, KDCUPD transfers the user data as if the value YES was specified for all the TRANSFER operands.

ASYNLPAP=

- | | |
|-----|--|
| YES | All asynchronous messages that have not yet been output to partner applications (distributed processing via LU6.1 / OSI TP) are also transferred.

The following applies to UTM cluster applications:
The parameter only applies to node updates. |
| NO | These messages are not transferred. |

ASYNTACS=

YES All background jobs not yet processed, including time-driven jobs and all asynchronous jobs with their data, are transferred. In addition, all messages in all TAC queues are transferred.

Messages from the dead letter queue are taken on regardless of whether the original destination still exists in the new generation or DEAD-LETTER-Q=YES was generated for TACs.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

**CAUTION!**

In the case of wraparound queues, the messages which were transferred first are lost and replaced by the most recently transferred messages when the queue level is reached. No warning message is issued.

NO These jobs and data are not transferred.

Neither queued messages nor open asynchronous messages are transferred.

ASYNTERM=

YES All asynchronous messages not yet output to LTERM partners are transferred, including time-driven messages.

NO These messages are not transferred.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

DIALOGS=

YES The data to dialog services is transferred.
If the service is open, this includes LSSBs, KB and the last dialog message.
If the service is terminated, only the last logged dialog message is transferred.

The following applies to UTM cluster applications:

- In the case of a node update, the service-specific data of connection user IDs is taken over.
- In the case of a cluster update, the service-specific data of genuine user IDs is taken over.

NO No service-specific data is transferred.

B	LOCALE=	
B B B B	YES	KDCUPD transfers the current values of the locale of each UTM user (USER) to the new KDCFILE. The values can differ from generated values, e.g. if a user has changed his/her locale using the SIGN CL call in the application run.
B B		The following applies to UTM cluster applications: The parameter only applies to cluster updates.
B	NO	The locales of users are not transferred, i.e. the generated values apply.
	PASS=	
	YES	<p>Passwords are transferred from the old to the new KDCFILE. This applies to all USERS for whom a password was generated in the old and in the new KDCFILE. The following is also transferred (if generated):</p> <ul style="list-style-type: none"> – the remaining validity period of the password – the most recently used passwords, in other words the password history – the minimum period before the password can next be changed <p>In the case of users for whom no password was defined in the old KDCFILE (in contrast to the new file), the new password is retained. If no password is generated for a user in the new KDCFILE, any existing password in the old KDCFILE is not transferred.</p> <p>The following applies to UTM cluster applications: The parameter only applies to cluster updates.</p>
	NO	No passwords are transferred.
	PROG-VER=	
	YES	<p>The current version numbers of the load modules (BS2000), shared objects (Unix systems) or DLLs (Windows systems) are transferred to the new KDCFILE.</p> <p>The following applies to UTM cluster applications: The parameter only applies to node updates.</p>
	NO	The current version numbers are not transferred.
	QUEUES=	
	YES	<p>All temporary queues and the messages they contain are transferred from the old to the new KDCFILE.</p> <p>The following applies to UTM cluster applications: The parameter only applies to node updates.</p>
	NO	The temporary queues and the messages they contain are not transferred.

SOCKET-DIALOGS=

YES The data for dialog services started by socket partners is transferred. In the case of an open service, these are the LSSBs, KB and the most recent dialog message. In the case of a terminated service, it is the most recently saved dialog message.

The following applies to UTM cluster applications:

- In the case of a node update, the service-specific data of connection user IDs is taken over.
- In the case of a cluster update, the service-specific data of genuine user IDs is taken over.

NO The data is not transferred.

STORAGES=

YES All UTM secondary storage areas, i.e. GSSB, TLS and ULS are transferred.

The following applies to UTM cluster applications:

- In the case of a node update, the TLS areas are taken over.
- In the case of a cluster update, the GSSBs and the ULS areas are taken over.

NO The UTM secondary storage area are not transferred.

UPIC-DIALOGS=

YES The data for dialog services started by UPIC clients is transferred. In the case of an open service, these are the LSSBs, KB and the most recent dialog message. In the case of a terminated service, it is the most recently saved dialog message.

The following applies to UTM cluster applications:

The parameter only applies to cluster updates.

NO The data is not transferred.

8.5 KDCUPD runtime log and messages

The update tool KDCUPD creates a runtime log that contains the following important information in addition to the parameters specified:

- Specifications on the data that was transferred.
- Specifications on the data that could not be transferred (these messages are marked with *).
- Brief information on the page pool usage

KDCUPD compares the generation of the old and new KDCFILE.

As a result of these checks, KDCUPD can reject transfer of individual items of user data because they are incompatible with the generation options of the new KDCFILE.

It is also possible for KDCUPD to reject transfer completely because individual generation options of the old and new KDCFILE differ so significantly that it would not be possible to start the application with the new KDCFILE and the transferred data (see also [page 582](#)).

The runtime log is output to SYSOUT and SYSLST or to *stdout* and *stderr* by default. The output can be controlled using the LIST statement.

The KDCUPD messages are listed in the openUTM manual “Messages, Debugging and Diagnostics”. The causes of error and the actions to be taken in response to the UTM message are described where necessary.

X/W Under Unix systems and Windows systems KDCUPD uses the NLS message catalog to output its messages.

Behavior in the event of errors

If an internal error occurs, KDCUPD creates a UTM dump (under Unix systems and Windows the dump is located in the DUMP subdirectory of the base directory).

This dump can be edited using the KDCDUMP editing tool (see the openUTM manual “Messages, Debugging and Diagnostics”).

B Under BS2000/OSD the process switch 3 is set if KDCUPD cannot terminate itself normally due to an error. Process switch 3 is also set when not all of the data could be transferred to the new KDCFILE because some generation components have been removed (see also message K266) although KDCUPD terminated itself normally.

B The process switch 3 is also set if KDCUPD could not run because an error occurred during checking the KDCFILES (see messages K258, K263 and K256).

Diagnostic documentation

If an error message is output in relation to the execution of KDCUPD, the following documentation should be supplied or at least saved:

- UTM dump, if one was created
In the event of a memory bottleneck, it may be the case that no dump file can be written.
X For UTM applications under Unix systems the core dump also must be logged.
- log of KDCUPD
- KDCDEF control statements for the old and new KDCFILE
(unless prohibited for data protection reasons)
- the old KDCFILE
- the new KDCFILE in the state before the KDCUPD run
(alternatively KDCDEF control statements)
- in the case of cluster updates:
the old cluster files and the new cluster files in their state before the KDCUPD run

9 Appendix

This chapter describes how to use code conversion tables and how to modify these tables.



Note that the use of code conversion only makes sense when transferring printable data and not binary data.

9.1 Code conversion tables in BS2000/OSD

B If communication from BS2000 to the partners is conducted via the TCP/IP protocol, then
B the partner usually uses the ASCII or ISO 8859-1 character set while BS2000 generally
B uses the EBCDIC character set. To make communication from BS2000 to this partner
B simple in spite of this, openUTM provides automatic code conversion. You can activate
B automatic code conversion during KDCDEF generation for specific partners using the
B MAP= operand in the PTERM or TPOOL statement. You can use different conversion tables
B for the actual conversion.

B Conversion tables

B Up to four different conversion tables can be used for the conversion. Table 1 already
B contains data upon delivery. It is intended for 7-bit ASCII (the eight bit is ignored in ASCII
B code). Tables 2, 3 and 4 are free for use and can be supplied with data by the user.
B All 8 bits are significant in these tables.

B The tables are defined in the KDCEA Assembler module. The KDCEA source can be found
B in the SYSLIB.UTM.061.EXAMPLE library. If a table is to be changed, then this module
B must be modified. The module must then be reassembled and linked to the application
B program via the INCLUDE-MODULES statement. This statement must act on the
B SYSLNK.UTM.061 library before the RESOLVE-BY-AUTO statement.

B The table to be used is specified during the generation of the partner in the PTERM or
B TPOOL statement in the MAP= operand.

9.2 Code conversion tables in Unix systems and Windows systems

X/W When exchanging messages of a UTM application with a partner application, openUTM has
X/W the ability to automatically execute a ASCII-EBCDIC code conversion. You can activate
X/W automatic code conversion during KDCDEF generation for specific partners using the
X/W MAP=SYSTEM operand in the KDCDEF statements PTERM, TPOOL, OSI-CON and
X/W SESCHA.

X/W openUTM uses a standard table for conversion. This is in the *utmpath* under `src/kcsaeee.c`
X/W (Unix systems) or `src\kcsaeee.c` (Windows systems).

9.2.1 Modifying the code table in Unix systems

X You can modify this standard table for UTM applications in Unix systems. Proceed as
X follows to do this:

- X 1. Copy the file `kcsaeee.c` into a separate directory.
- X 2. Modify the table as desired.
- X 3. Compile the modified source file.
- X 4. Link the work process by specifying the `.o` object before the `libwork.a` library. If you
X link with the `libwork.so` library, then you must regenerate this library beforehand. The
X script `utmpath/shsc/stat2dyn` is provided for this purpose.

9.2.2 Modifying the code table in Windows systems

W The code conversion tables are contained in the library `utmconvt.dll`. `utmconvt.dll` is
W located in the same directory as `libwork.dll`.

W You can modify these conversion tables to suit your own needs by modifying the supplied
W source files and then creating a modified `utmconvt.dll`.

W Source files for creating new code tables

W The directory `utmpath\src` contains the following source files which you may need to modify:

W ● `kcsaeea.c` and `kcxaent.c`

W `kcsaeea.c` contains the conversion tables for previous openUTM-Server versions and
W is used in the `utmconvt.dll` supplied with the product.

W `kcxaent.c` contains the complete tables for conversion between the Windows character
W set and EBCDIC.

W These files are C source files and each contain two character arrays with 256 elements.
W One array is used for conversion from ASCII to EBCDIC and the other array is used for
W conversion from EBCDIC to ASCII (see the example below).

W ● `utmconvt.def`

W Definition file containing EXPORT statements. It is not necessary to modify this file.

W ● `utmconvt.rc` and `resource.h`

W Resource files containing version and copyright information. This information is
W displayed when you right-click the DLL file and choose *Properties*. These files need not
W necessarily be linked in.

W **Modifying the library utmconvt.dll****W** Three steps are necessary to convert the library `utmconvt.dll`:

- W** 1. Modify the code table to suit your requirements (as necessary). Do this by editing
W `kcsaeea.c` or `kcxaent.c` with a text editor.
W If you only wish to use the conversion table `kcxaent.c` in place of the standard
W conversion table `kcsaeea.c`, you can omit step 1.

W *Example***W** You wish to incorporate German umlauts in the conversion table `kcsaeea.c`. Proceed
W as follows for the letter 'Ä':

- W** a) Find out the code for 'Ä'. 'Ä' has the code 'X'C4' (= decimal 196) in ISO 8859-1 and
W the code 'X'63' (= decimal 99) in EBCDIC.DF.04-1.
- W** b) Change the value of `kcsaebc[196]` from `0xff` to `0x63` (ASCII-to-EBCDIC conversion)
- W** c) Change the value of `kcseasc[99]` from `0x1a` to `0xc4` (EBCDIC-to-ASCII conversion)

W Proceed in the same way for the remaining German umlauts.**W** 2. Start Microsoft Visual C++ Developer Studio and proceed as follows:

- W** ● Create a new project with the name `utmconvt` in the directory `UTMPATH\utmconvt`.
W The project must be of the type *Visual C++/Win32/Win32 Project/Type: DLL*.
- W** ● Add the following files to the project:
 - W** – The code table you modified (either `kcsaeea.c` or `kcxaent.c`, see 1.),
 - W** – `utmconvt.def`
 - W** – and, if required, `utmconvt.rc`.
- W** ● From this project, create the library `utmconvt.dll`.
- W** ● Close Developer Studio

W 3. Replace the old library `utmconvt.dll` with the new library:

- W** ● First back up the library under a different name, so that you can access it again if
W anything goes wrong.
- W** ● Copy the new `utmconvt.dll` to the directory which contains the UTM library
W `libwork.dll` (this is generally `utmpath\ex`). Make sure that the original `utmconvt.dll`
W is actually replaced by the new `utmconvt.dll`.

W The new conversion library is ready for use.

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 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/OSD)

During execution of a *UTM application*, UTM events which are of relevance in terms of security are 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/OSD)

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/OSD)

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 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 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 in 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/OSD)

If the product *XHCS* (eXtended Host Code Support) 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 and can be changed dynamically by the administrator (while the application is running). The configuration is stored in the *KDCFILE*.

Also:

configuration

The process of defining the configuration of the UTM application. A distinction is made between *static* and *dynamic configuration*.

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.

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 systems/Windows systems)

A dialog terminal process connects a terminal of a Unix system or a 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/OSD)

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 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.

generation

Static configuration of a *UTM application* using the UTM tool KDCDEF and creation of an application program.

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 Sun Microsystems 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).

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, *openUTM* 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 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 during *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.

main process (Unix systems / Windows systems)

Process which starts the *UTM application*. It starts the *work processes*, *printer processes*, *network processes* and the *timer process* and monitors the *UTM application*.

main routine KDCROOT

See *KDCROOT*.

mapped host name

Mapping of the partner application's 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, and transfer of the message is still guaranteed, irrespective of whether a network connection is available at the time or not. In openUTM there are *UTM-controlled queues* and *service-controlled queues*.

message router (BS2000/OSD)

Device in a central host or a communication computer which distributes queued input messages to different *UTM applications* which can be located on different computers. The message router also allows you to work with *multiplex connections*.

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/OSD)

Special method of connecting 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 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.

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 globally unique (i.e. throughout the world). An object identifier comprises a sequence of integers which represent a path in a tree structure.

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.

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.

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.

postselection (BS2000/OSD)

Selection of logged UTM events from the SAT logging file which are to be evaluated. Selection is carried out using the SATUT tool.

predialog (BS2000/OSD)

Request from a terminal user to the data communication system to establish a *virtual connection* to the *application*. The predialog is unnecessary if the application requests the establishment of a virtual connection.

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/OSD)

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/OSD).

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 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. In BS2000/OSD 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/OSD)

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.

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 server 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-on 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/OSD)

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 (KDACS)

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)

An 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 the transport system interface (e.g. CMX 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 client

The designation for openUTM clients with the UPIC carrier system.

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.

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 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 2Kb or 4Kb. In *standalone UTM applications*, the size of a UTM page on generation of the UTM application can be set to 2K or 4K. The size of a UTM page in a *UTM cluster application* is always 4K. 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 systems, you must set *UTMPATH* before a UTM application is started. On Windows systems, *UTMPATH* is set on installation.

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/OSD)

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 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, the 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.

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.

work process (Unix systems / Windows systems)

A process within which the *services* of a *UTM application* run.

WS4UTM

WS4UTM (**Web**Services 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/OSD)

XHCS (Extended Host Code Support) is a BS2000/OSD 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/OSD)
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 Systems
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000/OSD)
DB	Database
DC	Data Communication
DCAM	Data Communication Access Method

DCOM	Distributed Component Object Mode
DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000/OSD)
DMS	Data Management System
DNS	Domain Name Service
DP	Distributed Processing
DSS	Terminal (Datensichtstation)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans TM
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GSSB	Global Secondary Storage Area
HIPLEX [®]	Highly Integrated System Complex (BS2000/OSD)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IFG	Interactive Format Generator
ILCS	Inter-Language Communication Services (BS2000/OSD)
IMS	Information Management System (IBM)
IPC	Inter-Process Communication
IRV	International Reference Version
ISO	International Organization for Standardization
J2EE	Java 2 Enterprise Edition Technologie
JCA	Java Connector Architecture
JDK	Java Development Kit
JEE5	Java Enterprise Edition 5.0
KA	KDCS Application Area
KB	Communication Area
KBPRG	KB Program Area

KDCS	Compatible Data Communication Interface
KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000/OSD)
LSSB	Local Secondary Storage Area
LU	Logical Unit
MIGRAT	Migration Program
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000/OSD)
NB	Message Area
NEA	Network Architecture for TRANSDATA Systems
NFS	Network File System/Service
NLS	Native Language Support
OCX	OLE Control Extension
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
PDN	Program System for Remote Data Processing and Network Control
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/OSD)
RTS	Runtime System
SAT	Security Audit Trail (BS2000/OSD)

Abbreviations

SECOS	Security Control System
SGML	Standard Generalized Markup Language
SLU	Secondary Logical Unit
SM2	Software Monitor 2 (BS2000/OSD)
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/OSD (Task Privileged)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Non-Privileged Function State in BS2000/OSD (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

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 access 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



PDF files of all openUTM manuals are included on the Enterprise DVD with open platforms and on the openUTM WinAdmin DVD (for BS2000/OSD).

All manuals are available as online manuals, see <http://manuals.ts.fujitsu.com>.

openUTM documentation

openUTM
Concepts and Functions
User Guide

openUTM
Programming Applications with KDCS for COBOL, C and C++
Core Manual

openUTM
Using openUTM Applications under BS2000/OSD
User Guide

openUTM
Using openUTM Applications under Unix Systems and Windows Systems
User Guide

openUTM
Administering Applications
User Guide

openUTM
Messages, Debugging and Diagnostics in BS2000/OSD
User Guide

openUTM
Messages, Debugging and Diagnostics in Unix Systems and Windows Systems
User Guide

openUTM (BS2000/OSD, Unix systems, Windows NT)
Creating Applications with X/Open Interfaces
Core Manual

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
Online description and online help system

openUTM, openUTM-LU62
Distributed Transaction Processing
between openUTM and CICS, IMS and LU6.2 Applications
User Guide

openUTM (BS2000/OSD)
Programming Applications with KDCS for Assembler
Supplement to Core Manual

openUTM (BS2000/OSD)
Programming Applications with KDCS for Fortran
Supplement to Core Manual

openUTM (BS2000/OSD)
Programming Applications with KDCS for Pascal-XT
Supplement to Core Manual

openUTM (BS2000/OSD)
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/OSD environment



Most of these manuals are available in printed form which must be paid and ordered separately at <http://manualshop.ts.fujitsu.com>

AID (BS2000/OSD)
Advanced Interactive Debugger
Core Manual
User Guide

BCAM (BS2000/OSD)
BCAM Volume 1/2
User Guide

BINDER (BS2000/OSD)
User Guide

BS2000/OSD
Executive Macros
User Guide

BS2000/OSD-BC
BLSSERV
Dynamic Binder Loader / Starter
User Guide

DCAM (BS2000/OSD)
COBOL Calls
User Guide

DCAM (BS2000/OSD)
Macros
User Guide

DCAM (BS2000/OSD)
Program Interfaces
Description

FHS (BS2000/OSD)
Format Handling System for openUTM, TIAM, DCAM
User Guide

IFG for FHS
User Guide

FHS-DOORS (BS2000/OSD,MS-Windows)
Graphical Interface for BS2000/OSD Applications
User Guide

HIPLEX AF (BS2000/OSD)
High-Availability of Applications in BS2000/OSD
Product Manual

HIPLEX MSCF (BS2000/OSD)
BS2000 Processor Networks
User Guide

IMON (BS2000/OSD)
Installation Monitor
User Guide

MT9750 (MS Windows)
9750 Emulation under Windows
Product Manual

OMNIS/OMNIS-MENU (BS2000/OSD)
Functions and Commands
User Guide

OMNIS/OMNIS-MENU (BS2000)
Administration and Programming
User Guide

OMNIS-MENU (BS2000/OSD)
User Guide

OSS (BS2000/OSD)
OSI Session Service
User Guide

RSO (BS2000/OSD)
Remote SPOOL Output
User Guide

SECOS (BS2000/OSD)
Security Control System
User Guide

SECOS (BS2000/OSD)
Security Control System
Ready Reference

SESAM/SQL (BS2000/OSD)
Database Operation
User Guide

openSM2 (BS2000/OSD)
Software Monitor
Volume 1: Administration and Operation

TIAM (BS2000/OSD)
User Guide

UDS/SQL (BS2000/OSD)
Database Operation
User Guide

Unicode in BS2000/OSD
Introduction

VTSU (BS2000/OSD)
Virtual Terminal Support
User Guide

XHCS (BS2000/OSD)
8-Bit Code and Unicode Support in BS2000/OSD
User Guide

Documentation for the Unix system environment

CMX V6.0 (Solaris)
Operation and Administration
User Guide

CMX V6.0 (Unix systems)
Operation and Administration
User Guide

CMX V6.0
Programming CMX Applications
Programming Guide

OSS (UNIX)
OSI Session Service
User Guide

PRIMECLUSTERTM
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

CPI-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>

Index

- 102
 - port number 153
 - 24-bit addressing mode 411
 - 31-bit addressing mode 411
 - 8-bit codes 237
 - 9001-3 printer 451
 - 9001-893 printer 451
 - 9011-18 printer 451
 - 9011-19 printer 451
 - 9012 printer 451
 - 9013 printer 451
 - 9022 printer 452
- A**
- abstract syntax 98
 - define 275
 - identify 528
 - ABSTRACT-SYNTAX 99, 249, 275, 286
 - access authorization 338
 - common memory pool 410
 - access list concept 219
 - access permissions
 - client/client programs 519
 - defining for user 534
 - access point 93
 - define (OSI TP) 277
 - generate 101
 - identify by AET 544
 - TNS entry 279, 282
 - access protection
 - LTERM pool 520
 - access rights
 - client (LTERM pool) 134
 - partner application 432
 - UTM terminal (KDCDEF) 360
 - access to UTM application 437
 - ACCESS-POINT 99, 249, 277
 - ACCOUNT 247, 283
 - accounting
 - CPU seconds 284
 - print output 284
 - record 284
 - weighting of partner application 350
 - accounting parameters
 - define 283
 - accounting phase 350, 498
 - accounting units
 - number 284
 - acknowledgment procedure
 - printer 179
 - activate
 - SAT logging 396
 - address
 - generate (OSI TP) 423
 - OSI TP access point 280
 - OSI TP partner 93
 - address format 281
 - BCAMAPPL 298
 - CON 316
 - OSI-CON 425, 456
 - address information
 - CMX 110
 - ADD-SPOOL-CHARACTER 174
 - ADD-SPOOL-FORM 174

- administration
 - print job queue 179
 - printer queue 358, 442
- administration authorization
 - partner application (LU6.1) 345
 - partner application (OSI TP) 432
 - transaction code 488
 - user 537
- administration authorizations 149
- administration call
 - asynchronous, define destination 379
 - result 379
- administration journal 622
- administration TAC
 - generate 484
- administrative data 49
 - initialize 49
 - KDCFILE 49
 - modification by transactions 52
 - update 54
 - UTM-F application 49
 - UTM-S application 49
- administrative data (KDCFILE) 31
- administrative information
 - assign process 121
- administrator
 - define 543
- ADMINISTRATOR=*ADMINISTRATOR 173
- AEQ 96, 97, 426
 - OSI-LPAP 428
- AES methods 231
- AET 96, 278, 426, 544
- alias LTERM 358
- ALTERNATE-LIBRARIES 340
- ALTERNATE-LIBRARIES=NO 202
- ALTERNATE-LIBRARIES=YES 202
- announce asynchronous message 514
- APDN 452
- APPLI 452, 453
- application
 - define default locale 385
 - define locale 132
- application context 96
 - define 285, 426
 - generate 100
 - OSI-LPAP 426
- application entity qualifier 96, 97, 426
- application entity title 96, 278, 426, 544
 - generate 100
- application message module 188
- application name
 - generate 291
- application process title 96, 426
 - UTMD 544
- application properties 33
 - generate 368
- APPLICATION-CONTEXT 99, 249
- APT 96, 426
 - OSI-LPAP 429
 - UTMD 544
- AREA 246, 288
- areas
 - KDCFILE 45
- ASCII code 238
- ASCII/EBCDIC 422, 447, 472
- assign
 - client/printer to LTERM partner 447
 - session characteristics 346
- assigning roles 222
- assigning user roles 222
- association (OSI TP) 95
- asynchronous administration
 - FPUT 53
- asynchronous message
 - character set used 241
 - display 357
 - maximum number 363
 - output without prior announcement 196
 - time-driven 49
- asynchronous processes
 - maximum number 373
- asynchronous processing 373
- asynchronous service 49, 499
- asynchronous tasks
 - maximum number 373
- ASYNTASKS 210, 373

AT-PRINTER-START 173
 authorization key
 semaphore 396
 authorization profile
 LTERM partner 129
 autolink function 202, 340, 341
 automatic
 connection 145
 automatic connection setup 170
 automatic KDCSIGN 365
 automatic mode
 printer 179
 automatic service restart 364, 542
 automatic sign-on 145
 automatically establishing a connection 442

B

BADTACS 484
 base name 46, 47
 KDCFILE 46, 383
 new KDCFILE (KDCUPD) 589
 BCAM 374
 BCAM application name 291
 BCAM generation 440
 distributed processing 109
 BCAM name 440
 BCAMAPPL 83, 130, 195, 247, 248, 291, 441
 for Clients 131
 BCAMAPPL name 573
 BCPMAP
 for clients 156, 160
 BCPMAP entries 109
 block length
 user data 387
 block size
 KDCFILE 45
 page pool 50
 blocked call
 wait time 548
 blocking call 503
 waiting time 391
 BLS interface
 using 339

BRETRYNR
 MAX 374
 BS2000/OSD
 minimal configuration 36
 BUNDLE 138
 bypass mode 178
 printer 178

C

cache
 (non-)resident 376
 define properties 375
 define size 375
 cache paging algorithm 376
 CAE specification 466
 call
 inverse KDCDEF 270
 KDCDEF 254, 588
 KDCDEF (BS2000/OSD) 254
 KDCDEF (Unix system) 255
 KDCDEF (Windows) 256
 KDCUPD (BS2000/OSD) 588
 KDCUPD (Unix system) 588
 KDCUPD (Windows) 588
 catalog entry
 KDCFILE 376
 catalog IDs
 node application 312
 CATID
 KDCFILE 601
 catid 46
 CCS name 239
 edit option 329, 361, 385, 520
 edit profile 242
 CERTIFICATE 533
 certificate 533
 character interpretation 280
 character set
 compatible 239
 extended 237, 240
 character set name 239
 format 242
 locale 240

- CID [170](#), [442](#)
 - printer [179](#)
- PRINTER, printertype, [454](#)
- client [128](#)
 - assign LTERM partner [129](#)
 - BCMAP entry for [156](#), [160](#)
 - connect [129](#)
 - connect via LTERM pool [133](#)
 - define [355](#), [364](#)
 - define (PTERM) [437](#)
 - logical [129](#)
 - multiple connections [134](#)
 - properties for physical [130](#)
 - secure [232](#)
 - status [455](#)
 - trusted [232](#)
 - TS application [453](#)
- client program, see openUTM client program
- client/server cluster, generating [163](#)
- close session [77](#)
- CLUSTER [247](#), [249](#), [250](#), [299](#)
- cluster
 - configuring a node application [310](#)
 - relocatable IP address [381](#)
- cluster administration journal [622](#)
- cluster configuration file [67](#)
 - generating [301](#), [417](#)
- cluster GSSB file [68](#)
 - generating [301](#), [417](#)
- cluster page pool
 - number of files [308](#)
 - size, warning level [308](#)
- cluster page pool files [68](#)
 - generating [301](#)
- cluster ULS file [68](#)
 - generating [301](#), [417](#)
- cluster update [590](#)
- cluster user file [302](#)
 - generating [301](#), [417](#)
 - user locales [309](#)
 - UTM cluster application [67](#)
- CLUSTER-FILEBASE
 - defining new base name (KDCUPD) [602](#)
 - KDCUPD statement [602](#)
- cluster-internal communication
 - communication end point [302](#)
 - port number [302](#)
- CLUSTER-NODE [247](#), [249](#), [250](#), [310](#)
- CMX
 - providing address information [110](#)
- code conversion
 - ASCII/EBCDIC [422](#)
 - for TS applications [152](#)
- code types [350](#)
- coded character set name [239](#)
- CODED-CHARACTER-SET [240](#)
- combination
 - multiplex connection - direct connection [197](#)
- ComfoTRAVEL [550](#)
- comment line
 - insert [460](#)
 - KDCDEF [258](#), [259](#)
- commit functional unit [278](#)
- common memory pool
 - generate [205](#), [410](#)
 - generate size [411](#)
 - shared code [205](#)
 - size [205](#)
- communication
 - with partner [100](#)
- communication area
 - define fill character [377](#)
 - define length [382](#)
- communication partner
 - define (PTERM) [437](#)
 - define type [450](#)
 - type [457](#)
 - UPIC-L/UPIC-R [453](#)
- compatible character sets [239](#)
- compiler [434](#), [435](#)
- CON [84](#), [248](#), [274](#), [313](#)
 - changing dynamically [571](#)
- configuration [33](#)
 - change dynamically [569](#)
 - define [32](#), [254](#)
 - example application [550](#)
 - store configuration information [31](#)
- configuration data (KDCROOT) [31](#)

- configuration file
 - UTM cluster application 67
- configuration information
 - store 33
- configure
 - node application 310
 - UTM cluster application 299
- CONNECT 170, 195
- connect
 - client program, multiple 134
- connect client 129
- connection
 - automatic 145
 - define (LU6.1) 313
 - define name (OSI TP) 430
 - generate (OSI TP) 101
 - parallel (OSI TP) 95
 - transport protocol 450
- connection bundle 137
- connection control time 545
- connection module 234
- connection point, logical 128
- connection request 454
- connection request time 171, 378
- connection setup
 - automatic 442
 - global timeout 545
 - RSO printer 168
- connection shutdown
 - confirmation (MUX) 196
- connection user ID 144
 - restart 150
- CONNECT-MODE 134
- CONN-USERS 132
- CONRTIME 171, 378
- CONSOLE 188
- contention loser 471
 - LU6.1 77
 - OSI TP 95
- contention winner 471, 472
 - LU6.1 77
 - OSI TP 95
- continuation lines
 - KDCDEF control statements 258
- control
 - application load 377
 - KDCDEF run 416
 - SAT logging 468
 - utilization 132
- control identification
 - printer 179
- control sign-on procedure 478
- control statements
 - create 43, 268
 - enter 254
 - format 258
 - KDCDEF 254
 - KDCUPD 600
 - order 257
- control_statements_file 318, 417
- CONTWIN 471
- conversation exit 491
- conversion 152
 - to native TCP-IP connections 521
- conversion file 123
- conversion tables 613
- copy KDCFILE 45
- CPI-C program unit
 - blocking calls 492, 503
 - TAC 484, 489
- CPU seconds
 - weighting 284
- CPU time
 - define 498
- create
 - control statements 43, 268
 - control statements for object types 319
 - new KDCFILE 587
- CREATE-CONTROL-STATEMENTS 43, 247, 318
- cssname 240
- CTERM 168

D

- data access control [33](#), [219](#), [337](#)
 - define for services [102](#)
 - for service-controlled queues [224](#)
 - transaction code [491](#)
 - using encryption functions [232](#)
 - with distributed processing [226](#)
- data areas
 - define [288](#)
 - shareable [206](#)
- data conversion ASCII/EBCDIC [472](#)
- data protection [147](#)
- data security [48](#)
- data transfer
 - application context [285](#)
 - transfer syntax (BER) [98](#)
- DATABASE [246](#)
- database connection
 - define [321](#)
- database key [489](#)
- database linking [235](#)
 - define [234](#)
 - multi-instance mode [236](#)
- DATABASE statement [234](#)
- database system [234](#)
 - define [321](#), [466](#)
 - KDCUPD [582](#)
 - maximum number [321](#)
- DB [321](#)
- DCAM application [452](#)
- dead letter queue
 - number of messages [379](#)
- DEFAULT [246](#), [324](#)
- default language environment [132](#)
- default system code [239](#)
- default user character set [240](#)
- default values
 - application-specific [324](#)
 - define [324](#)
- define
 - abstract syntax [275](#)
 - accounting parameters [283](#)
 - alias LTERM [358](#)
 - application context [285](#)
 - define (cont.)
 - base name for KDCFILE [383](#)
 - client [355](#)
 - configuration [254](#)
 - database linking [234](#)
 - database system [321](#)
 - default values [324](#)
 - edit options [328](#)
 - event exits [334](#)
 - format handling system [336](#)
 - function keys [473](#)
 - key set [337](#)
 - language environment [240](#)
 - locale [240](#)
 - logical connection (LU6.1) [313](#)
 - LPAP bundle [367](#)
 - LPAP partner [344](#)
 - LTERM group [358](#)
 - LTERM pool [511](#)
 - master LPAP [367](#)
 - master LPAP of an LU6.1-LPAP bundle [366](#)
 - master LTERM [357](#)
 - multiplex connections [194](#)
 - OSI-LPAP partner [426](#)
 - parallel connections (LU6.1) [291](#)
 - physical clients/printers [437](#)
 - physical printer [169](#)
 - printer [168](#), [355](#)
 - printer control LTERM [168](#), [179](#), [358](#)
 - process values [368](#)
 - program unit [434](#)
 - Resource Manager [466](#)
 - RSO buffer size [176](#)
 - session name [347](#)
 - shared objects [476](#)
 - SM2 data supply [397](#)
 - start format [358](#)
 - TAC [483](#)
 - TAC classes [500](#), [575](#)
 - transfer syntax [528](#)
 - user ID [530](#)
 - user message module [187](#), [406](#)
 - define character set
 - edit option [329](#)

- define locale
 - TPOOL 519
- define names 260
- delete
 - dynamically 269, 274
- DES encryption 229
- DES methods 231
- destination_process_name 471
- device definition 173
 - example 174
- dialog control 548
- dialog message 49
 - character set used 241
- dialog service 499
- dialog TACs 497
- DISCONNECT PENDING 196
- DISCONNECT=*YES 173
- disk partition
 - estimate 60
- distributed processing
 - define global values 544
 - define partner application (LU6.1) 344
 - generate (OSI TP) 99
 - generate connection (LU6.1) 313
 - generate connection (OSI TP) 420
 - generate session name (LU6.1) 347
 - local TAC 348
 - OSI TP 93
- documentation
 - summary 15
- DPUT
 - LTERM bundle 137
 - LTERM group 141
- DPUTLIMIT1 379
- DPUTLIMIT2 380
- DSS 3270 (IBM) 451
- DSS 9748 451
- DSS 9749 451
- DSS 9750 451
- DSS 9751 451
- DSS 9752 451
- DSS 9753 451
- DSS 9754 451
- DSS 9756 451
- DSS 9763 451
- DSS X28 (TELETYPE) 451
- DSS X28 (VIDEO) 451
- dual-file operation
 - KDCFILE 45, 383
 - raw-device 62
- dynamic
 - deletion 269, 274
 - entering objects 461, 569
 - object entry 268
- E**
- EBCDIC 238
- EBCDIC character sets 238
- EBCDIC/ASCII conversion
 - native TCP-IP communication 521
- EDIT 247, 328
 - define CCS name 329
 - define character set 329, 361, 385, 520, 535
- edit options
 - define 328
- edit profile 328
 - character set name 242
- EJECT 246, 258, 332
- encryption 228
 - passwords 230
 - transaction code 490
- encryption level 231
 - generation 231
 - specifying for clients 443, 515
- encryption methods 229
- ENCRYPTION-LEVEL
 - PTERM 443, 490, 515
- END 246, 333
- enter
 - control statements 254
 - objects, dynamically 461
 - RSO device manager 173
 - UTM objects, dynamically 570
- ENTRY 234, 246, 247
- error documentation 611
- event exit SERVICE 491
- event exits 203
 - define 334

event services 203, 334
event-specific preselection 468
example
 example generation ComfoTRAVEL 550
 generate locale 242
example generation 550
execution time
 time-driven job 379
EXIT 334, 491
explicit connection user ID 144
extended character set 237

F

F keys 473
FHS-DOORS Front End 451
file that is global to the cluster
 lock 307
 lock request retry 307
FILEBASE
 node application 310
filebase
 derived files 46
filebase directory
 other files 47
fill character
 communication area/standard primary working
 area 377
follow-up program 491
FORMAT 335, 533
format
 KDCDEF control statements 258
 names (KDCDEF) 260
format handling system
 define 336
FORMMODE 335
FORMSYS 246, 336
FPUT 53
 LTERM bundle 137
 LTERM group 141
function keys
 define 473
functions
 KDCUPD 600

G

generate
 address (OSI TP) 423
 administration TAC 484
 application properties 368
 cluster configuration file 301, 417
 cluster GSSB file 301, 417
 cluster page pool files 301
 cluster ULS file 301, 417
 cluster user file 301, 417
 distributed processing (OSI TP) 93
 ID card check 532
 introduction 31
 KDCFILE 417
 load module (BLS) 339
 load modules 199
 logical connection (OSI TP) 420
 LU6.1-LPAP bundle 89
 multiplex connection 414
 OSI-LPAP bundle 106
 printer 168
 printer pool 178
 ROOT table source 417
 RSA keys 418
 RSO printer 172
 shareable modules 206
 shareable objects 206
 UTM and BCAM 109
 UTM application 32
 UTM cluster application 64
 UTM database linking 236
 UTM objects 34
 UTM-D application (LU6.1) 76
 UTM-D application (OSI TP) 93
generating
 LU6.1 connection 81
generating a UTM cluster application 64
generating distributed applications
 using WinAdmin 75
generation
 change 587
 client/server cluster 163
 maximum value exceeded 263
 terminate 263

- generation interface 32
- generation notes
 - LU6.1 protocol 78
- generation tool 245
- generation variants of UTM 371
- global application pool 205
- global properties
 - UTM cluster application 299
- global values
 - define 102
 - distributed processing 544
- Grace-Sign-On 479
- GROUP 142
- GSSB 49
 - define maximum number 381
- H**
- high transaction rate 57
- high-performance file 374
- Hiperfile 374
- host name
 - mapped 122
 - node application 311
 - real 122
 - UTM 122
- HOSTCODE 239
- HOSTNAME 381
- HP-UX 14
- hyphen
 - in names 261
- I**
- I/O behavior 57
- ID card check
 - generate 532
- ID card information 532
 - define length 376
- identifier
 - communication partner 433
- identify
 - printer 170
- idle
 - connection (OSI TP) 431
 - session 471
- IDLETIME 147, 431, 471
- IMON installation path
 - SESAM/SQL 322
 - UDS/SQL 322
- implicit connection user ID 144
- INFO LO 237
- INFORMIX 466
- INIT PU 237
- initialize
 - administrative data 49
- INPUT 335
- input files
 - KDCDEF 274
- input/output
 - weighting 284
- insert
 - comment line 460
- instance (OSI) 93
- intelligent terminal 452
- internationalization 237
 - application 406
- introduction
 - generation 31
- inverse KDCDEF 43, 268, 318
 - call 270
 - generated files 271
 - object types 268
 - start 270
- IP address
 - relocatable 381
- IPC 382
- IPCTRACE 382
- ISO 646 238
- ISO 8859 238
- ISO character sets 238
- IUTMDB 321
- J**
- job control 208
- job receiver confirmation
 - waiting time 546
- job receiver response
 - waiting time (KDCDEF) 354

- job-receiving services
 - maximum number 546
- job-specific SAT logging 495

- K**
- K keys 473
- K009 474
- K040 52
- K041 52
- K101 52
- K492 277
- KAA 267
- KAA size 267
- KAASHMKEY 382
- KCMF 173
- KCSMSGs 186, 189, 406
- KCSMSGSE 186, 406
- KDC Application Area 267
- KDCAPLKS 337
- KDCAPPL SPOOLOUT=ON 169
- KDCBADTC 210, 484
- KDCDEF
 - call 254, 588
 - call (BS2000/OSD) 254
 - call (Unix systems) 255
 - call (Windows) 256
 - comment line 258, 259
 - control statements 245
 - enter control statements 254
 - input files 417
 - inverse KDCDEF 271
 - logging 259
 - messages 567
 - name classes 265
 - name conventions 261
 - number of names 262
 - optimize input files 274
 - page feed 258
 - result 34, 267
 - terminate input 333
- KDCDEF control statements
 - continuation lines 258
 - format 258
- KDCDEF run
 - control 416
- KDCFILE 31, 33, 45, 267
 - administrative data 49
 - areas 45
 - base name 46
 - block size 45
 - catalog entry 376
 - CATID 601
 - check consistency 601
 - control data transfer 605
 - copy 45
 - create new 587
 - creating during operation 55
 - data transfer with KDCUPD 583
 - define base name 383
 - define new base name 601
 - dual-file operation 45, 383
 - filebase directory 47
 - generate 254
 - generate dual-file operation (KDCDEF) 376
 - generating 417
 - main file 58
 - number of files 58
 - page pool 49
 - performance enhancement 57
 - raw-device 59
 - restart area 52
 - specifying new base name 603
 - split 57, 383
 - update 577
 - updating for a UTM cluster application 590
 - UTM cluster application 70
- KDCINF PTERM 196
- KDCMMOD 186, 190
- KDCMSGTC 210, 484
- KDCMTXT 186
- KDCPTERM 196
- KDCROOT 31, 254
- KDCROOT main routine 31
- KDCS message area
 - define length 388
- KDCSGNTC 210, 484
- KDCSIGN 365, 376, 532

- KDCTXT 190
- KDCUPD 589
 - call 588
 - call (BS2000/OSD) 588
 - call (Unix system) 588
 - call (Windows) 588
 - check KDCFILE consistency 601
 - control data transfer 606
 - control runtime log 604
 - control statements 600
 - necessary steps 587
 - prerequisites 578
 - RSA key 580, 581
- Kerberos 537
 - generating system access control 243
- Kerberos dialog
 - performing 243, 359, 518
- Kerberos principal 530
- KERBEROS-DIALOG 359, 518
- key code 219, 337
 - define maximum value 384
- key set 337
 - assign to LTERM partner 360
 - changing dynamically 569
 - define 337
 - LTERM pool 134, 519
 - partner application 432
 - user 534
- KEYSET 134
- KSET 129, 220, 222, 247, 274, 337
 - changing dynamically 571
 - with inverse KDCDEF 268
- L**
- lang_id 186, 240
- language environment
 - application (default) 385
 - application-specific 241
 - define 240
 - LTERM partner-specific 241, 361, 519
 - LTERM-specific 129, 135
 - standard 241
 - user-specific 241, 535
- language identifier
 - locale 240
- language-specific message modules 187
 - assign 406
- layer (OSI) 93
- LEADING-SPACES 384
- LEASY 234, 321
- length
 - buffer 53
- level of complexity
 - password 538
- LIB 187, 234
- LINEMODE 335
- linked operation 235
- Linux distribution 14
- listener ID 120, 296
 - MAX 389
- LISTENER-ID
 - cluster communication 307
- LISTENER-PORT 131, 132
 - ACCESS-POINT 279, 422
 - BCAMAPPL 292, 296
 - CON 315
 - PTERM 446
- load
 - control 377
- load mode 200
 - load module 341
 - shared object 476
- load module (BLS)
 - autolink function 340
 - define load mode 341
 - generate 339
 - load mode 341
 - version number 343
- load modules
 - generate 199
 - libraries 201
- LOAD-MODULE 246, 247, 339
- local 347
- local application pool 205
- local service name
 - changing dynamically 569

- local session name
 - define 347
- LOCALE 129, 132, 135, 187, 188, 241
 - default language environment 385
 - LTERM partner-specific 361
- locale 237, 240
 - define 240
 - generation example 242
- local-session-name 76
- LOCK 129, 134
- lock
 - transaction code 496
 - user ID 543
- lock code 219, 338, 349, 573
 - LTERM partner 362
 - LTERM pool 520
 - remote service 348
 - transaction code 491
- lock/key code concept 219, 360, 491
- locked resource
 - waiting time 394
- log
 - page feed 332
- LOGACKWAIT 171
- logging
 - activate 396
 - KDCDEF 259
- logical access point
 - client/printer 355
 - partner application 344
 - partner application (OSI TP) 426
- logical client 129
- logical connection
 - define (LU6.1) 313
 - define (OSI TP) 420
 - reestablish 171, 378
- logical connection point 128
- logical print acknowledgement 171, 386
- long-term storage area
 - user-specific 529
- LPAP 248
 - LU6.1-LPAP bundle 89
- LPAP bundle 367
 - OSI TP protocol 105
- LPAP bundles, LU6.1 protocol 88
- LPAP name 315
- LPAP partner
 - assign session characteristics 346
 - define 344
- LPUT buffer 386
- LPUT data 386
- LPUT message 387
- LPUTLTH 387
- LSES 85, 248, 274, 347
 - changing dynamically 571
- LSSB 49
 - define maximum number of 387
- LTAC 99, 226, 248, 249, 274, 348
 - changing dynamically 571
 - for inverse KDCDEF 268
- LTERM 131, 134, 170, 221, 223, 247, 355
 - LTERM bundle 138
 - LTERM group 142
 - reserve table locations 462
- LTERM bundle
 - LTERM statement 138
 - PTERM statement 139
- LTERM group 140, 358
 - LTERM statement 142
 - PTERM statement 142
- LTERM partner 128, 129, 138, 142, 573
 - administer message queue 168
 - assign 133
 - assign client/printer 447
 - authorization profile 129
 - define locale 361
 - define locale for LTERM pool 519
 - define properties 355
 - defining access permissions 360
 - language environment 241
 - LTERM group 140, 358
 - LTERM pool 133
 - message queue 169, 363
 - name 129, 168, 357
 - of an LTERM pool 520
 - predefine 355
 - specific language environment 241
 - system access control 129

- LTERM pool
 - access permissions 519
 - connecting clients 135
 - define 511
 - define communication partner type 524
 - define LTERM partners 520
 - LTERM partner names 134
 - maximum number of clients 522
 - message queue 525
 - names of LTERM partners 520
 - number of clients 134
 - properties 133
 - system access control 134
 - types 133
- ltermname 129, 168
- ltermprefix 520
- LU6.1
 - generation 81
 - generation notes 78
- LU6.1 protocol 76
- LU6.1-LPAP bundle 88
 - generating 89
 - making an LPAP a slave LPAP 345
 - standalone application with UTM cluster
 - application 90
- LU6.2 applications 75
- M**
- magnetic strip card 532
- main file
 - KDCFILE 58
- main routine KDCROOT 254
- mapped host names 122
- master LPAP
 - define 367
 - defining for an LU6.1-LPAP bundle 366
- master LTERM 137
 - specify 357
- MASTER-LU61-LPAP 89, 248, 366
- MASTER-OSI-LPAP 99, 106, 367
- MAX 120, 131, 185, 246, 247, 250, 368
- MAX OSI-SCRATCH-AREA 100, 249
- MAX OSISHMKEY 100, 249
- MAX statement 129, 171
- maximum length
 - physical messages 132, 171
- maximum waiting time 147
 - sign on to common memory pool 388
- MAXSES 195
- MESSAGE 187, 190, 246, 406, 409
- message
 - leading blanks 384
 - message definition file 186, 189
 - message destination 188
 - user-specific 191
 - message display, with WinAdmin 191
 - message distribution 192
 - basic principle 193
 - message encryption
 - transaction code 490
 - message module 406, 409
 - create 186
 - English 186
 - Standard 188
 - user-specific 186
 - message queue 363
 - LTERM partner 169, 363
 - LTERM pool 525
 - maximum number of messages 363
 - OSI-LPAP partner 432
 - message router
 - functions 194
 - message tools 190
 - messages
 - KDCDEF 567
 - KDCUPD 610
 - metasyntax 28
 - minimal configuration
 - BS2000/OSD 36
 - modify
 - KDCFILE administrative data 49
 - page pool size 50
- MODIFY-SPOOL-DEVICE 173
- MODULE 187, 246, 247
- MPOOL 410
- MP-WAIT 388
- MSCF 625
- MSG-DEST 191

- MSGTAC 484
- multi-instance mode
 - database linking 236
- multi-lingual message module 406
- multilingualism 237
 - UTM program units 237
- multiple sign-ons 146, 480
- multiplex 194
- multiplex connection 197
 - avoid data jam 198
 - define 194
 - generate 192, 414
 - PTERM 450
- multiplex function 192
- multi-step transaction, wait time 400
- multi-threaded 120, 389
- multi-threaded network access 120
 - ACCESS-POINT 279
 - BCAMAPPL 296
 - MAX 368
- multi-threaded network connection
 - BCAMAPPL 307
- MUX 247, 414
 - asynchronous message 196
 - confirm connection shutdown 196
 - statement 195
- MUX connection
 - statistics 197
- N**
- name
 - format (KDCDEF) 260
 - maximum number (KDCDEF) 262
 - partner application 315
 - remote application (LU6.1) 315
 - session characteristics 470
- name classes, KDCDEF 265
- name conventions 261
- NEABT 450, 524
- NET-ACCESS 388
- network access 120, 388
- NLS standard message catalog 189
- node application 63
 - availability 304
 - bound service 72
 - catalog IDs 312
 - communication retries 304
 - configuring 310
 - emergency command 305
 - failure command 307
 - FILEBASE 310
 - generating with provisional values 70
 - host name 311
 - reply time 304
 - warm start 309
- node update 590
- node-bound service 303
- nonprivileged subsystems 204
- notational conventions 28
- NRCONV 389
- NUMBER 134
- number
 - asynchronous services 373
 - tasks 501
- O**
- object identifier 285
- object protection 219
- object types
 - inverse KDCDEF 268
- OMNIS 192, 414
- OMNIS-MENU 195
- ONCALL 342, 476
- open
 - session 77, 472
- OpenCPIC
 - data access protection 227
- OpenCPIC client 143
- openUTM client program 452
 - multiple connections via LTERM pool 515
- OPTION 34, 45, 246, 416
- OPTION DATA 274
- OPTION DATA=control_statements_file 271, 417
- ordering application 226
- OSI terms 93

- OSI TP 93
 - generate connection 420
- OSI TP access point 422
- OSI TP partner
 - address 93
 - generate 101
- OSI-CON 99, 249, 420
- OSI-LPAP 99, 249, 426
 - OSI-LPAP bundle 106
- OSI-LPAP bundle 105
 - generate 106
 - standalone application with UTM cluster application 105
- OSI-LPAP partner 128
 - assign partner application 420
 - define 426
 - maximum number asynchronous messages 432
 - replacement connections 422
- output messages 49

- P**
- padding_count_time 471
- page feed
 - log 332
- page pool 45, 49
 - block size 50
 - define properties 390
 - define size 386
 - estimate size 50
 - size 47, 50
 - warning levels 51
- page pool size
 - KDCDEF 363
 - modify 50
- page pool utilization
 - define warning levels 390
- PAM read/write jobs 394
- parallel connections
 - define (LU6.1) 291
 - general OSI TP 95
 - generate (LU6.1) 313
 - generate (OSI TP) 101
 - generate number of (OSI TP) 430
- PARTNER 188
- partner application
 - access authorization 345
 - access rights 432
 - address (OSI TP) 423
 - administration authorization 345
 - assign 313
 - define (LU6.1) 344
 - define local TAC 348
 - define LPAP partner 344
 - logical access point 344
 - name 348
 - name (LU6.1) 315
- partner type 450, 524
- partner/server application 227
- password
 - blanked-out input 536
 - define level of complexity 538
 - encrypted 230
 - user ID 536
- password history 481
- PDN 452
- PDN application 452
- performance 57, 147, 372
 - CONTWIN 471
 - monitoring, generate 397
- performance enhancement
 - KDCFILE 57
- PGPOOL 171
- PGPOOLFS 58
- PGWT
 - TAC class 492, 503
 - waiting time 391
- physical clients
 - properties 130
- physical message
 - maximum length 401
- PLEV 168
- PLU 77, 472
- pool
 - global application 205
 - local application 205
- port number 102 110, 153
 - Unix systems and Windows systems 110

- prefix 46
- prerequisites
 - dynamic entry of objects 569
 - KDCUPD 578
- preselection
 - event-specific 468
 - predefine values 468
- presentation layer 95
- presentation selector 280
- primary logical unit 77, 472
- PRINCIPAL 537
- PRINCIPAL-LTH 392
- print acknowledgment 386
- print job
 - weighting 284
- print job queue 179
 - administer 179
- print process 442
- print_level_number 362
- PRINTER 453
- printer
 - activate for openUTM 177
 - assign LTERM partner 170
 - bypass mode 178
 - connection setup 362
 - control identification (CID) 179
 - define 168, 355, 365
 - define (PTERM) 437
 - generate 168
 - identify 170
 - logical 168
 - OLTP interface 172
 - status 455
- printer (RSO)
 - release in the event of error 178
- printer control 442
- printer control LTERM 179, 442
 - define 168, 179, 358
- printer group 169
- printer ID 170, 442
- printer information (RSO)
 - query 177
- printer pool 179
 - generate 178
- printer queue
 - administer 358, 442
- printer sharing 169, 362
- printer status (RSO) 177
- printer type 170
- printertype 454
- priority control 208
 - generate 505
- PRISMA 234, 321
- private key (RSA) 229
- process
 - maximum number 399
 - number per TAC class 500
- process limitation
 - TAC classes 208
- process values
 - define 368
- processor
 - partner application 316
- processor name 316, 415, 449, 523
- PROGRAM 246, 247, 434
 - reserve table locations 463
- program
 - assign transaction code 492
- program exchange 476
 - binder 436
 - BLS 341
 - generate (BLS) 339
- program unit 574
 - define 434
 - weighting 350, 498
- program wait 503
- PRONAM 134, 172
- properties
 - clients and printers 33
 - LTERM pools 133
 - partner applications 33
 - physical clients 130
 - TACs 33
 - UTM application 33
- properties of UTM application
 - generate 368
- protecting objects 219
- PROTECT-PW 538

- PTCTIME 546
- PTERM 247, 437
- LTERM bundle 139
 - LTERM group 142
 - reserve table locations 463
- ptermname 130, 169, 172
- PTYPE 131, 135, 170, 172, 450, 524
- LTERM bundle 139
 - LTERM group 142
- public key (RSA) 229
- PUTMMUX connection 194
- PUTMMUX protocol 194
- Q**
- QAMSG 169
- query
- printer information (RSO) 177
- QUEUE 184, 247
- queue level
- USER queue 540
- queue_level_number 363, 525, 540
- queues
- asynchronous message 363
 - defined with TAC statement 181
 - service-controlled 181
 - temporary 181
 - user-specific 181
- R**
- raw-device, KDCFILE 59
- README files 20
- real host name 122
- real user ID 144
- restart 150
- RECBUF 87
- RECBUFFS= 58
- receiving application 227
- recipient_TPSU-title 350
- Red Hat 14
- reestablish
- logical connection 171, 378
- reference code 239
- regeneration, recommendations 273
- relocatable 381
- relocatable IP address 381
- REMARK 246, 258, 460
- remote application, see partner application
- remote session name
- define 347
- REMOTE-BUFFER-SIZE 172
- remote-session-name 76
- replacement connection
- (OSI TP) 102
- RESERVE 246, 247, 461
- reserve
- empty table locations 571
 - table locations 33
 - table locations for UTM objects 461
- reserve node application
- generating 70
- reserved names
- KDCDEF 260
- RESET RSO printer 173
- RESET=*YES 173
- Resource Manager 235
- define 466
- RESOURCE WAIT 394
- RESTART
- LTERM bundle 138
- restart
- connection user IDs 150
 - real user ID 150
- restart area 47, 52
- define size 393
 - size 47, 53
 - split (KDCDEF) 393
- restart information 52
- result
- inverse KDCDEF 271
 - KDCDEF 34
 - KDCDEF run 267
- RFC1006 281
- RFC1006 connection
- BCAM 109
- RFC1006 link
- example 163, 165
- RMXA 246, 466
- RMXA statement 235

- ROOT [34](#), [246](#), [467](#)
 - generating the table source [417](#)
 - ROOT table source
 - define name [467](#)
 - RSA [229](#)
 - RSA key
 - length [231](#)
 - transfer to new KDCFILE [580](#), [581](#)
 - RSA key pair [229](#)
 - creation [233](#)
 - RSA keys
 - generate [418](#)
 - RSET [547](#)
 - RSO [172](#), [173](#)
 - device entry [174](#)
 - RSO buffer size
 - define [176](#)
 - RSO device manager
 - enter [173](#)
 - RSO printer
 - generate [172](#), [453](#)
 - RTS modules [341](#)
 - runtime parameters [49](#)
- S**
- SAT logging
 - activate [396](#)
 - control [468](#)
 - TAC-specific [495](#)
 - user-specific [543](#)
 - SATSEL [247](#), [468](#)
 - secure client [232](#)
 - selector
 - format [95](#)
 - OSI TP [93](#)
 - semaphores [396](#)
 - service
 - bound to node application [72](#)
 - protected [222](#)
 - service access point [93](#)
 - generate [101](#)
 - service program (remote)
 - lock code [349](#)
 - TAC name [350](#)
 - service restart [364](#), [533](#), [542](#)
 - service stacking [389](#)
 - service-controlled queue [49](#)
 - data access control [224](#)
 - number of redeliveries [185](#)
 - wait time for reading [185](#)
 - SESAM [321](#)
 - SESAM/SQL [234](#)
 - SESCHA [82](#), [248](#), [470](#)
 - session [193](#), [194](#)
 - close [77](#)
 - define characteristics (LU6.1) [470](#)
 - define session name (LU6.1) [347](#)
 - duration [77](#)
 - idle [471](#)
 - name [76](#)
 - open [77](#), [472](#)
 - open, global timeout [545](#)
 - reserve, waiting time [353](#)
 - session characteristics
 - assign [346](#)
 - session layer [95](#)
 - Session Manager [193](#)
 - OMNIS [414](#)
 - session selector [280](#)
 - session setup [193](#)
 - SFUNC [247](#), [473](#)
 - shareable data area [206](#)
 - shareable objects, generate [206](#)
 - shareable program units [204](#)
 - shareable programs
 - nonprivileged subsystems [204](#)
 - shared code
 - common memory pool [205](#)
 - shared memory [382](#)
 - shared memory segment
 - authorization key [375](#)
 - SHARED OBJECT [246](#), [248](#), [476](#)
 - shared objects
 - define [476](#)
 - file name [477](#)
 - version identifier [477](#)
 - SHARED-OBJECT [436](#)
 - SHUT [335](#)

- SIGN CL [237](#)
 - SIGNON [247](#), [478](#), [484](#)
 - sign-on
 - automatic [145](#)
 - SINGLE-THREADED [389](#)
 - single-threaded network access [121](#)
 - size
 - cluster page pool [308](#)
 - KAA [267](#)
 - page pool [47](#), [50](#)
 - page pool (KDCDEF) [363](#)
 - restart area [47](#), [53](#)
 - size monitoring
 - SYSLOG [398](#)
 - slave LTERM [137](#), [357](#)
 - SM2 data supply
 - define [397](#)
 - SNA sessions [76](#)
 - SOCKET [453](#)
 - providing address information [118](#)
 - socket application [452](#), [453](#)
 - socket connections
 - EBCDIC/ASCII conversion [521](#)
 - Solaris [14](#)
 - SPEC=C [236](#)
 - specify input files
 - KDCDEF run [417](#)
 - split
 - KDCFILE [57](#)
 - SPOOL parameter file [173](#)
 - standalone UTM application [13](#)
 - standard language environment [241](#)
 - standard message module
 - KCSMSGS [189](#)
 - standard primary working area
 - define fill character [377](#)
 - define length [398](#)
 - standard UTM message module [406](#)
 - START [335](#)
 - start
 - inverse KDCDEF [270](#)
 - KDCDEF [254](#)
 - start error KF58 [398](#)
 - start format [517](#), [533](#)
 - define [358](#)
 - user-specific [533](#)
 - START-PRINTER-OUTPUT [173](#), [177](#)
 - STARTUP [341](#), [476](#)
 - STATIC [342](#)
 - static generation, object components [573](#)
 - STATION [188](#)
 - statistics
 - on MUX connections [197](#)
 - stdin [255](#), [256](#)
 - steps
 - KDCFILE update [585](#)
 - updating the KDCFILE for a UTM cluster application [590](#)
 - store
 - configuration information [31](#), [33](#)
 - subsystems
 - nonprivileged [204](#)
 - SUSE [14](#)
 - SYSDTA [254](#)
 - SYSLINE [188](#)
 - SYSLNK
 - DATABASE [322](#)
 - SYSLOG [398](#)
 - size monitoring [398](#)
 - SYSLST [188](#)
 - SYSLST log
 - generate page feed [332](#)
 - SYSMSH.UTM.060.MSGFILE [186](#)
 - SYSOUT [188](#)
 - system access control [148](#)
 - generating with Kerberos [243](#)
 - LTERM partner [129](#)
 - LTERM pool [134](#)
 - using encryption functions [231](#)
- T**
- table locations
 - for UTM objects [461](#)
 - reserve [571](#)
 - reserved [33](#)

- TAC [182](#), [220](#), [223](#), [224](#), [246](#), [247](#), [483](#)
 - administration authorization [488](#)
 - define [483](#)
 - reserve table locations [463](#)
- TAC class control [49](#)
- TAC classes [497](#)
 - define [500](#), [575](#)
 - setting priorities [505](#)
- TAC name
 - service program (remote) [350](#)
- TAC properties
 - generate [483](#)
- TAC queue [181](#), [182](#)
 - generate [499](#)
- TACCLASS [247](#), [500](#), [505](#)
- TAC-PRIORITIES [247](#), [505](#)
- TAC-specific SAT logging [495](#)
- TACUNIT [498](#)
- task
 - number of [503](#)
- TASKS [502](#)
 - maximum number [399](#)
- TCB entries
 - define [509](#)
- TCBENTRY [246](#), [498](#), [509](#)
- TCP/IP
 - port 102 [110](#)
- temporary queue [181](#), [184](#)
- terminal ID [316](#)
- terminal identifier [526](#)
 - communication partner [455](#)
- TERMINAL WAIT [400](#)
- terminal-specific long-term storage area [510](#)
- TERMWAIT [147](#)
- terr_id [186](#), [240](#)
- territorial identifier
 - locale [240](#)
 - LTERM partner [361](#)
- TESTMODE [382](#)
- time to think
 - user [400](#)
- time-driven job
 - time of execution [379](#)
- timeout
 - connection setup (OSI TP) [545](#)
 - opening of session (LU6.1) [545](#)
- timer
 - input from dialog partner [445](#), [518](#)
- TLS [49](#), [510](#)
- TLS block [510](#)
 - define name [510](#)
- TNS mode [113](#)
 - converting to KDCDEF [114](#)
- TPOOL [247](#), [511](#)
- TPOOL statement [133](#)
- T-PROT [131](#), [132](#), [281](#)
- TPSU-title [96](#)
- TRACE=YES [178](#)
- transaction code [575](#)
 - data access control [491](#)
 - define for partner application [348](#)
 - define name [488](#)
 - lock [496](#)
 - lock code [491](#)
 - remote application [348](#)
- transaction rate [57](#)
- transaction-oriented processing [277](#)
- transactions
 - default space requirements in restart area [53](#)
- transfer
 - RSA key to new KDCFILE [580](#), [581](#)
 - user data [275](#)
- transfer message
 - number of attempts [374](#)
- transfer syntax [98](#)
 - define [528](#)
- TRANSFER-SYNTAX [100](#), [249](#), [528](#)
- transport confirmation [386](#)
- transport connection [76](#), [93](#), [194](#)
 - define [313](#)
- transport connection to LU6.1
 - changing dynamically [569](#)
- transport layer [95](#)
- transport protocol [450](#), [524](#)
- transport selector [281](#)
- transport system application [128](#), [452](#)

- travel reservation system
 - example generation 550
 - TRMSGLTH 132, 171, 172
 - trusted client 232, 445
 - TS application 128, 437
 - code conversion 152
 - multiple connection via LTERM pool 515
 - T-selector
 - generate partner application 424, 425
 - TSEL-FORMAT 131, 132
 - TSOS 178
 - TTY 453
 - tuning
 - KDCFILE 57
 - TYP 234
 - type
 - communication partner 450
 - communication partner for LTERM pool 524
 - type identifier of database system 234
- U**
- UDS 321
 - UDS/SQL 234
 - UDTAC 286
 - UDTCCR 287
 - UDTDISAC 286
 - UDTSEC 287
 - ULS 49, 247, 529
 - underscore
 - in names 261
 - Unix platform 14
 - update
 - administrative data 54
 - KDCFILE 577
 - KDCFILE (UTM cluster application) 590
 - UPIC client program
 - UPIC-L/UPIC-R 453
 - UPIC connection
 - signing off a user 481
 - UpicB.ocx 25
 - UPIC-R 452
 - define openUTM client program 437
 - USAGE 130, 131, 169
 - LTERM bundle 139
 - LTERM group 142
 - USAGE=O 170
 - USER 181, 221, 223, 225, 247, 530
 - reserve table locations 464
 - user
 - maximum number 377
 - sign-on check 536
 - user data
 - current 49
 - encode for transfer 285
 - encrypted 230
 - transfer 98, 275
 - user data (KDCFILE) 31
 - user file
 - user locales 309
 - UTM cluster application 67
 - user ID 33, 144, 365, 576
 - define 530
 - lock 543
 - LTERM partner 365
 - password 536
 - user locales
 - taking over specifications into the cluster user file 309
 - user log file
 - buffered records 49
 - define name 383
 - dual-file operation 401
 - user message modules 188
 - define 187, 406
 - define language environment 406
 - USER queue
 - queue level 540
 - USER queues 181
 - user services protocol 450, 524
 - USER-DEST number (user-specific message destinations) 191
 - USERFORM 335
 - userid 46

- user-specific
 - language environment 241
 - SAT logging 543
 - start format 533
 - user-specific long-term storage area 529
 - user-specific message destinations 191
 - user-specific queues 181
 - USP (UTM socket protocol) 151
 - utilization
 - control 132
 - UTM accounting, see accounting
 - UTM application
 - access to 437
 - define name 368
 - generate 32
 - UTM cache 50
 - UTM cluster application 13, 63
 - checking the availability of nodes 304
 - cluster administration journal 622
 - cluster configuration file 67
 - cluster user file 67
 - communication retries 304
 - configuring 299
 - defining global properties 299
 - emergency command 305
 - failure command 307
 - generating 64
 - KDCDEF statements 69
 - KDCFILE 70
 - LU6.1-LPAP bundle 90
 - OSI-LPAP bundle 105
 - reply time 304
 - using global memory areas 71
 - UTM cluster files
 - generating 417
 - UTM database linking
 - generate 236
 - utm file directory/msgdescription 189
 - UTM host names 122
 - UTM messages
 - user-specific adaptation 186
 - UTM object
 - generate 34
 - reserve table location 571
 - UTM object tables 33
 - UTM page 50
 - define size 374
 - UTM SAT administration authorization
 - partner application (LU6.1) 346
 - partner application (OSI TP) 432
 - transaction code 495
 - user 537
 - UTM SAT administration TACs
 - generate 486
 - UTM socket protocol (USP) 151
 - UTM user ID, see user ID
 - UTM_NET_HOSTNAME 125
 - UTM-C.CPMD 68
 - UTM-C.CPnn 68
 - UTM-C.GSSB 68
 - UTM-C.ULS 68
 - UTMD 99, 100, 248, 249, 544
 - UTM-D application
 - define global values 544
 - generate 76
 - utm-directory/sys/kcsmsgs.o 409
 - UTM-F application 372
 - administrative data 49
 - utmhostname 124
 - UTM-INFORMIX linking 235
 - utmnetm 121
 - UTM-ORACLE linking 235
 - utmpfad 123
 - UTM-S application 371
 - administrative data 49
- ## V
- variant numbers 238
 - version change
 - with KDCUPD 578
 - version identifier
 - shared objects 477
 - version number 201
 - load module (BLS) 343
 - virtual host name
 - utmhostname 124

virtual host names
 conversion file [123](#)
 UTM_NET_HOSTNAME [125](#)

VTSU code [176](#)

VTSU version [452](#)

VTSU-B [238](#)

W

waiting job
 job receiver confirmation [546](#)

waiting time
 job receiver response (KDCDEF) [354](#)
 locked resource [394](#)
 maximum [147](#)
 multi-step transaction [400](#)
 PGWT [391](#)
 reading from service-controlled queues [185](#)
 reserve session [353](#)

warm start
 node application [309](#)

warning level
 cluster page pool utilization [308](#)

warning level 1 [390](#)

WinAdmin
 generating distributed applications [75](#)
 message display [191](#)

Windows system [14](#)

X

XA [234](#), [321](#)

XA interface [235](#), [466](#)

xa_switch_t [466](#)

xa_switch_t structure [235](#)

XAPTP shared memory [100](#)

XAPTP shared memory segment [402](#)

XASWITCH [235](#)

XATMI program unit
 blocking calls [492](#), [503](#)
 TAC [484](#), [489](#)

XATMIAC [287](#)

XATMICCR [287](#)

XHCS [240](#)

XHCS definition of terms [238](#)

XHCS support [237](#)

