

English



FUJITSU Software

# openUTM V6.5

Creating Applications with X/Open Interfaces

User Guide

## **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](mailto: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](http://www.cognitas.de)

## **Copyright and Trademarks**

Copyright © 2017 Fujitsu Technology Solutions GmbH.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

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

---

# Contents

<b>1</b>	<b>Preface</b> . . . . .	<b>7</b>
<b>1.1</b>	<b>Summary of contents and target group</b> . . . . .	<b>9</b>
<b>1.2</b>	<b>Summary of contents of openUTM manuals</b> . . . . .	<b>10</b>
1.2.1	openUTM documentation . . . . .	10
1.2.2	Documentation for the openSEAS product environment . . . . .	14
1.2.3	Readme files . . . . .	15
<b>1.3</b>	<b>Changes in openUTM V6.5</b> . . . . .	<b>16</b>
1.3.1	New server functions . . . . .	16
1.3.2	Discontinued server functions . . . . .	23
1.3.3	New client functions . . . . .	24
1.3.4	New functions for openUTM WinAdmin . . . . .	25
1.3.5	New functions for openUTM WebAdmin . . . . .	25
<b>1.4</b>	<b>Notational conventions</b> . . . . .	<b>26</b>
<b>2</b>	<b>Communicating via X/Open interfaces</b> . . . . .	<b>27</b>
<b>2.1</b>	<b>The “Distributed Transaction Processing” reference model</b> . . . . .	<b>28</b>
<b>2.2</b>	<b>Integrating the X/Open interfaces</b> . . . . .	<b>29</b>
<b>2.3</b>	<b>CPI-C , XATMI and TX under openUTM</b> . . . . .	<b>31</b>
<b>2.4</b>	<b>Linking server/server with CPI-C and XATMI under openUTM</b> . . . . .	<b>32</b>
<b>2.5</b>	<b>Linking client/server with CPI-C and XATMI under openUTM</b> . . . . .	<b>33</b>
<b>2.6</b>	<b>Fields of application for CPI-C, XATMI, TX and KDCS</b> . . . . .	<b>34</b>
<b>2.7</b>	<b>Files and libraries for the X/Open interfaces</b> . . . . .	<b>35</b>

<b>3</b>	<b>X/Open CPI-C interface</b>	<b>37</b>
<b>3.1</b>	<b>The X/Open interface CPI-C</b>	<b>37</b>
3.1.1	Definition of terms	38
3.1.2	Communication partners of a CPI-C application under openUTM	40
3.1.3	Linking server/server with CPI-C	42
3.1.4	Sample application – flowchart	44
<b>3.2</b>	<b>CPI-C characteristics and functions in openUTM</b>	<b>46</b>
3.2.1	Conversation characteristic conversation_type	46
3.2.2	Conversation characteristics for addressing	47
3.2.3	Send-receive mode and send control	53
3.2.4	Multiple conversations in one CPI-C program	54
3.2.5	Conversation characteristic sync_level	56
3.2.6	Maximum message length	57
3.2.7	Converting characteristics and user data	57
3.2.7.1	Converting characteristics	57
3.2.7.2	Converting user data	57
3.2.8	States of a conversation under openUTM	64
<b>3.3</b>	<b>CPI-C in openUTM</b>	<b>66</b>
3.3.1	Supported CPI-C calls	66
3.3.2	Restrictions in conversations via the LU6.1 and UPIC protocol	69
3.3.3	openUTM-specific special features of CPI-C calls	70
3.3.4	Interaction with the TX interface	79
3.3.5	Behavior when non-supported CPI-C calls are used	82
3.3.6	Process or task switching	82
3.3.7	Programming rules	83
<b>3.4</b>	<b>Creating a CPI-C application</b>	<b>95</b>
3.4.1	Compiling and linking a CPI-C application on Unix, Linux and Windows systems	95
3.4.2	Compiling and linking a CPI-C application on BS2000 systems	96
3.4.3	Generating a CPI-C application	97
<b>3.5</b>	<b>Error diagnosis in CPI-C programs</b>	<b>100</b>
3.5.1	Controlling the trace	100
3.5.2	Name of the trace file	101
3.5.3	Contents of the trace file	102

---

<b>4</b>	<b>X/Open interface XATMI</b>	<b>103</b>
<b>4.1</b>	<b>Linking client/server</b>	<b>104</b>
4.1.1	Default server	105
<b>4.2</b>	<b>Communication paradigms</b>	<b>106</b>
<b>4.3</b>	<b>Typed buffers</b>	<b>110</b>
<b>4.4</b>	<b>Program interface</b>	<b>114</b>
4.4.1	XATMI functions	114
4.4.2	Particularities of XATMI calls	117
4.4.3	Data transfer to the service function	117
4.4.4	Events and error handling	118
4.4.5	Creating typed buffers	119
4.4.6	Characteristics of XATMI in openUTM	121
<b>4.5</b>	<b>Configuring</b>	<b>122</b>
4.5.1	Creating the local configuration file	122
4.5.2	The xatmigen utility	128
4.5.3	KDCDEF generation	131
4.5.3.1	Generating TACs for which PGWT ist permitted	131
4.5.3.2	Number of sessions (LU6.1) or associations (OSI TP)	131
4.5.3.3	Defining the services offered	131
4.5.3.4	Defining the services used	133
4.5.3.5	Example of requester generation	133
<b>4.6</b>	<b>Creating XATMI applications</b>	<b>137</b>
4.6.1	Include files and COPY elements	137
4.6.2	Linking the application on Unix, Linux and Windows systems	138
4.6.3	Linking the application on BS2000 systems	138
<b>4.7</b>	<b>Environment or job variables for XATMI</b>	<b>139</b>
4.7.1	Environment variables on Unix, Linux and Windows systems	139
4.7.2	Setting job variables on BS2000 systems	140
<b>4.8</b>	<b>Error diagnostics in XATMI programs</b>	<b>141</b>
4.8.1	Controlling the trace	141
4.8.2	Name of the trace file	142
<b>4.9</b>	<b>Interaction with the TX interface</b>	<b>143</b>
<b>4.10</b>	<b>Messages</b>	<b>144</b>
<b>4.11</b>	<b>T.61 character set</b>	<b>148</b>

<b>5</b>	<b>X/Open TX interface</b>	<b>149</b>
<b>5.1</b>	<b>transaction_control characteristic</b>	<b>149</b>
<b>5.2</b>	<b>TX interface calls under openUTM</b>	<b>150</b>
5.2.1	Particularities of TX calls under openUTM	150
<b>5.3</b>	<b>Interaction with the CPI-C interface</b>	<b>151</b>
<b>5.4</b>	<b>Interaction with the XATMI interface</b>	<b>153</b>
<b>5.5</b>	<b>Examples for the use of the TX interface</b>	<b>154</b>
<b>5.6</b>	<b>Creating an application with TX calls</b>	<b>156</b>
<b>5.7</b>	<b>Diagnosing errors in TX calls</b>	<b>157</b>
5.7.1	Controlling the trace	157
5.7.2	Name of the trace file	158
	<b>Glossary</b>	<b>159</b>
	<b>Abbreviations</b>	<b>197</b>
	<b>Related publications</b>	<b>203</b>
	<b>Index</b>	<b>211</b>

---

# 1 Preface

Today, vast quantities of data are used and analyzed to support corporate decision-making. Supplied by a wide range of networked users, devices, and “things”, the huge volume of data poses a major challenge to the IT infrastructure. Since these data quantities are going to continue growing, efficient transport, processing, and storage are also required. And, to be able to exploit the potential of digitalization, there also has to be a means of integrating large numbers of new applications at short notice.

In essence, therefore, modern IT infrastructure is required to

- Enable flexible, gradual scalability – without limits for the foreseeable future
- Provide server, storage, and network resources with self-service portals and automated solutions
- Enable flexible settlement models, such as consumption-based pricing
- Allow the integration of additional options for adjustments based on customer-specific requirements – such as different security requirements, etc.

To meet these challenges, Fujitsu offers an extensive portfolio of innovative enterprise hardware, software, and support services within the environment of our enterprise mainframe platforms, and is therefore your

- Reliable service provider, giving you longterm, flexible, and innovative support in running your company's mainframe-based core applications
- Ideal partner for working together to meet the requirements of digital transformation
- Longterm partner, thanks to high levels of investment in the continuous adjustment of modern interfaces required by a modern IT landscape with AI applications, robotics, and IoT, etc.

With openUTM, Fujitsu provides you with a thoroughly tried-and-tested solution from the middleware area.

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

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

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

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

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



## 1.1 Summary of contents and target group

The manual “Creating Applications with X/Open Interfaces” describes aspects that are specific to the X/Open interfaces CPI-C, TX and XATMI within openUTM applications. It is intended for all who want to use these interfaces for programming UTM applications.

This manual is designed as a supplement to the following four X/Open specifications dealing with “Distributed Transaction Processing” (please refer to the section [“Related publications” on page 203](#) for exact titles):

- “Reference Model Version 2”
- “The CPI-C Specification, Version 2”
- “The XATMI Specification”
- “The TX (Transaction Demarcation) Specification”.

Readers are expected to be familiar with these X/Open specifications and with openUTM; no description of the syntax of the individual calls is provided.

Chapter 2, which immediately follows this preface, provides a brief introduction to reference model and interfaces developed by X/Open and describes how this model fits in to the openUTM environment.

Chapters 3, 4 and 5 explain what you will need to know when working with the interfaces n CPI-C, XATMI and TX under openUTM.

The detailed reference section at the back of the manual - including a glossary, abbreviations, a list of related publications and a keyword index - is intended to help you get the most out of this manual.



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

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

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

## 1.2 Summary of contents of openUTM manuals

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

### 1.2.1 openUTM documentation

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

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

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

<http://manuals.ts.fujitsu.com>

On this site, enter the search term “openUTM V6.5” in the **Search by product** field to display all openUTM manuals of version 6.5.

The following sections provide a task-oriented overview of the openUTM V6.5 documentation. You will find a complete list of documentation for openUTM in the chapter on related publications at the back of the manual.

#### Introduction and overview

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

#### Programming

- You will require the **Programming Applications with KDCS for COBOL, C and C++** manual to create server applications via the KDCS interface. This manual describes the KDCS interface as used for COBOL, C and C++. This interface provides the basic functions of the universal transaction monitor, as well as the calls for distributed processing. The manual also describes interaction with databases.
- You will require the **Creating Applications with X/Open Interfaces** manual if you want to use the X/Open interface. This manual contains descriptions of the openUTM-specific extensions to the X/Open program interfaces TX, CPI-C and XATMI as well as

notes on configuring and operating UTM applications which use X/Open interfaces. In addition, you will require the X/Open-CAE specification for the corresponding X/Open interface.

- If you want to interchange data on the basis of XML, you will need the document entitled **openUTM XML for openUTM**. This describes the C and COBOL calls required to work with XML documents.
- For BS2000 systems there is supplementary documentation on the programming languages Assembler, Fortran, Pascal-XT and PL/1.

### **Configuration**

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

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

In addition, it also shows you how to transfer important administration and user data to a new KDCFILE using the KDCUPD tool. You do this, for example, when moving to a new openUTM version or after changes have been made to the configuration. In the case of UTM cluster applications, it also indicates how you 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 UTM Applications** manual for the relevant operating system (BS2000 or Unix, Linux and Windows systems). This describes how to link and start a UTM application program, how to sign on and off to and from a UTM application and how to replace application programs dynamically and in a structured manner. It also contains the UTM commands that are available to the terminal user. Additionally, those issues are described in detail that need to be considered when operating UTM cluster applications.

### **Administering applications and changing configurations dynamically**

- The **Administering Applications** manual describes the program interface for administration and the UTM administration commands. It provides information on how to create your own administration programs for operating a standalone UTM application or a UTM cluster application and on the facilities for administering several different applications centrally. It also describes how to administer message queues and printers using the KDCS calls DADM and PADM.

- If you are using the graphical administration workstation **openUTM WinAdmin** or the Web application **openUTM WebAdmin**, which provides comparable functionality, then the following documentation is available to you:
  - A **description of WinAdmin** and **description of WebAdmin**, which provide a comprehensive overview of the functional scope and handling of WinAdmin/WebAdmin. These documents are shipped with the associated software and are also available online as a PDF file.
  - The respective **online help systems**, which provide context-sensitive help information on all dialog boxes and associated parameters offered by the graphical user interface. In addition, it also tells you how to configure WinAdmin or WebAdmin in order to administer standalone UTM applications and UTM cluster applications.



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

### Testing and diagnosing errors

You will also require the **Messages, Debugging and Diagnostics** manuals (there are separate manuals for Unix, Linux and Windows systems and for BS2000 systems) to carry out the tasks mentioned above. These manuals describe how to debug a UTM application, the contents and evaluation of a UTM dump, the behavior of openUTM 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 product **openUTM-JConnect** shipped with **BeanConnect** is supplied with the software. This documentation consists of Word and PDF files that describe its introduction and installation and of Java documentation with a description of the Java classes.

- The **BizXML2Cobol** manual describes how you can extend existing COBOL programs of a UTM application in such a way that they can be used as an XML-based standard Web service. How to work with the graphical user interface is described in the **online help system**.
- If you want to provide UTM services on the Web quickly and easily then you need the manual **WebServices for openUTM**. The manual describes how to use the software product WS4UTM (WebServices for openUTM) to make the services of UTM applications available as Web services. The use of the graphical user interface is described in the corresponding **online help system**.

### **Communicating with the IBM world**

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

### **PCMX documentation**

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

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

## 1.2.2 Documentation for the openSEAS product environment

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

### Integrating Java EE application servers and UTM applications

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

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

### Connecting to the web and application integration

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

The manuals will also be supplemented by JavaDocs.

### 1.2.3 Readme files

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

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

#### *Information on BS2000 systems*

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

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

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

#### *Additional product information*

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

#### *Readme files on Unix and Linux systems*

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

#### *Readme files on Windows systems*

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

## 1.3 Changes in openUTM V6.5

The following sections provide more details about the changes in the individual functional areas.

### 1.3.1 New server functions

#### Long computer names of up to 64 characters

openUTM supports computer names that can be up to 64 characters in length. This means it is no longer necessary to map the name from the UTM generation to a real long name using configuration or mapping files.

The following interfaces have been changed to support this:

- Generation

*All systems:*

- KDCDEF statements CON, OSI-CON, PTERM and TPOOL:  
For PRONAM= or OSI-CON NETWORK-SELECTOR=, a name of up to 64 characters in length can be specified for the partner computer.

*BS2000 systems:*

- KDCDEF statements CON:  
The LISTENER-PORT= operand can also be specified on BS2000 systems.

*Unix, Linux and Windows systems:*

- KDCDEF statement CLUSTER-NODE:  
For HOSTNAME= and VIRTUAL-HOST=, a name of up to 64 characters in length can be specified.
- KDCDEF statement MAX:  
For HOSTNAME=, a name of up to 64 characters in length can be specified.

- Programming

- KDCS call INFO SI:
  - To receive the available information in full, value 180 must be specified in the KCLA field.
  - New fields KCHSTNML and KCPRONML for returning long computer names.
- KDCS call INIT PU:
  - The interface version has been increased to 6.
  - To receive the available information in full, value 356 must be specified in the KCLI field.



- New fields KCHSTNML and KCPRONML for returning the long computer names.
- KDCS call SIGN ST:
  - The interface version has been increased to 4.
  - To receive the available information in full, value 180 must be specified in the KCLA field.
  - The node name in the KCCLNODE field can be up to 64 characters in length.
- KDCADMI administration interface
  - New structure *kc\_long\_triple\_str* in the identification area to support long computer names. This structure must be used for all objects except KC\_MUX. As a result, the data structure version of KDCADMI has also been changed to version 10 (*version\_data* field in the parameter area).
  - New fields for the long computer names:

Data structure	Field name(s)
<i>kc_cluster_node_str</i>	<i>hostname_long</i> and <i>virtual_host_long</i>
<i>kc_con_str</i> <i>kc_lses_str</i> <i>kc_lterm_str</i> <i>kc_pterm_str</i> <i>kc_tpool_str</i>	<i>proname_long</i>
<i>kc_osi_con_str</i>	<i>network_selector_long</i>
<i>kc_max_par_str</i> <i>kc_system_par_str</i>	<i>hostname_long</i>

- KDCADM command interface
  - KDCINF command:  
If an object of the type CON, LSES, LTERM, OSI-CON, PTERM, or TPOOL contains a long computer name, the information is output in two screen lines.
  - Commands KDCLSES, KDCPOOL, KDCPTERM, and KDCSWTCH:  
If an object contains a long computer name, the information is output in two screen lines.
- Messages  
Message K037 is output every time the long computer name is converted to the short local name and vice versa.
- KDCNAMEINFO tool (Unix, Linux and Windows systems)  
The KDCNAMEINFO tool is provided for assigning IP address to computer name.

### Load-module version **\*HIGHEST-EXISTING (BS2000 systems)**

A UTM application on a BS2000 system can be generated in such a way that the highest version of the load module available in the library is always loaded, i.e., the version no longer has to be explicitly specified.

The following interfaces have been changed to support this:

- Generation

KDCDEF statement LOAD-MODULE:

New operand values VERSION = \*HIGHEST-EXISTING and \*UPPER-LIMIT.

\*UPPER-LIMIT is a synonym for the VERSION=@ value that was already possible.

- KDCADMI administration interface

Data structure *kc\_load\_module\_str*:

- *version* field: New values \*HIGHEST-EXISTING and \*UPPER-LIMIT. These can be specified when using KC\_MODIFY\_OBJECT to perform modifications.
- New field *version\_gen* for the generated version of the load module or shared object.



The *version\_gen* field is also filled on Unix, Linux and Windows systems.

- KDCADM command interface

- KDCINF command:

For the LOAD-MODULE object type, the generated version of the load module is output in VERSION (GENERATED).

- KDCPROG command:

For the VERSION parameter, \*HIGHEST-EXISTING and \*UPPER-LIMIT can also be specified.

### Password length of up to 16 characters

The password for a UTM user ID can be up to 16 characters in length.

- Generation

- KDCDEF statement USER:

For PASS=, a password of up to 16 characters in length can be specified.

For PROTECT-PW=, the maximum value for the minimum length has been increased to 16.

- Programming

- KDCS call SIGN ON, CK: In the KCLA field, value 16 must be specified.
- KDCS call SIGN CP: In the KCLA field, value 32 must be specified.

- KDCS call SIGN ST:
  - The interface version has been increased to 4.
  - To receive the available information in full, value 120 must be specified in the KCLA field.
  - New field KCRPASSL for returning the password from the UPIC protocol.
- KDCADMI administration interface
  - Data structure *kc\_user\_str* and *kc\_user\_dyn2\_str*:  
New fields *password16*, *protect\_pw16\_lth* and *pw\_encrypted*.
- KDCADM command interface
  - KDCUSER command  
For PASS=, a password of up to 16 characters in length can be specified.
- Program CALLUTM (BS2000 systems)
  - CREATE-CONFIGURATION statement  
For PASSWORD=, a password of up to 16 characters in length can be specified.

### Dead letter queue for LPAP and OSI-LPAP

Messages to LU6.1 or OSI-TP partners that are deleted because they could not be delivered due to a permanent error can now be saved in the dead letter queue.

The interfaces have been changed as follows to support this:

- Generation
  - KDCDEF statements LPAP and OSI-LPAP:  
The new operand DEAD-LETTER-Q= controls whether undeliverable messages that are deleted from their message queue are to be saved in the dead letter queue.
- Programming
  - For KDCS call DADM with KCOM=MV or MA (moving messages from the dead letter queue), an LPAP or OSI-LPAP partner can also be specified.
- KDCADMI administration interface
  - Data structures *kc\_lpap\_str* and *kc\_osi\_lpap\_str*: New field *dead\_letter\_q* for controlling the saving of undeliverable messages in the dead letter queue.

## Code conversion tables

For code conversion between ASCII-compatible codes and EBCDIC codes, openUTM provides tables for four code conversions on all platforms (two 8-bit conversions and two 7-bit conversions). By default, openUTM uses an 8-bit conversion between ISO8859-i and EBCDIC.DF.04-i on BS2000, Unix and Linux systems, and an 8-bit conversion between Windows-1252 and EBCDIC.DF.04-F on Windows systems.

The following interfaces have been changed to support this:

- Generation
  - KDCDEF statements PTERM and TPOOL:  
The operand values MAP=SYS1, SYS2, SYS3, and SYS4 can also be specified on Unix, Linux and Windows systems. The values may be specified only for transport system applications (partner type APPLI or SOCKET).
  - KDCDEF statements OSI-CON and SESCHA:  
New operand values MAP=SYS1, SYS2, SYS3, and SYS4 on Unix, Linux and Windows systems for the four conversions.
- Programming

The sources with the conversion tables are delivered on all platforms (BS2000: assembler module KDCEA; Unix, Linux and Windows systems: C source kcsaeea.c) and can be adjusted or replaced by custom tables on all platforms.
- KDCADMI administration interface
  - Data structures *kc\_pterm\_str* and *kc\_tpool\_str*:  
For the *map* field, additional values “1”, “2”, “3”, and “4” are available on Unix, Linux and Windows systems (for the four conversions). The values may be specified only for transport system applications (partner type APPLI or SOCKET).
  - Data structures *kc\_lpap\_str* and *kc\_osi\_con\_str*:  
For the *map* field, additional values “1”, “2”, “3”, and “4” are available on Unix, Linux and Windows systems (for the four conversions).

## User-specific error handling (Unix and Linux systems)

On Unix and Linux systems, a separate user signal routine can be created. This routine is called as soon as a signal occurs. It replaces the default error handling in openUTM (abnormal service termination + PENDER dump), i.e., it enables a service to be continued despite errors.

The following interfaces have been changed to support this:

- Programming
  - New functions `KCX_REG_SIGNAL_HANDLER` and `KCX_UN_REG_SIGNAL_HANDLER` for registering and deregistering the signal routine in openUTM.
  - New functions `KCX_SET_RELOAD_FLAG` for exchanging a work process after end-of-transaction, and `KCX_WRITE_DUMP` for creating a UTM dump without service cancellation.

The functions are available in COBOL and C.

### Access data for the XA database connection

- The access data for the XA database connection can (as was previously the case) be specified in the start parameters.
- The user name and the password for the XA database connection can be changed via administration (KDCADMIN):
  - Operation code `KC_MODIFY_OBJECT`:
    - New object type `KC_DB_INFO`
    - New values `KC_IMMEDIATE` and `KC_DELAY` in the *subopcode1* field for controlling the time of the password change.
  - Data structure *kc\_db\_info\_str*: New fields *db\_userid* and *db\_password*.
- The access data for the XA database connection can be transferred with KDCUPD:
  - TRANSFER statement: New operand `DB-CREDENTIALS=`.

### KDCUPD with changed number of databases

KDCUPD transfers all data even if the number of databases in the new generation increases. Prerequisites for this are that the openUTM version does not change and the sequence of the existing databases in the generation remains unchanged.

### Multi DBs on BS2000 systems

Up to three databases can be generated on BS2000 systems.

## Moving pending asynchronous messages for LTERM, LPAP, and OSI-LPAP bundles

For LTERM, LPAP, and OSI-LPAP bundles, pending asynchronous messages for slave LTERMS, slave LPAPs, or slave OSI-LPAPs can be automatically moved to a slave with an established connection after the waiting time has elapsed.

- Generation, KDCDEF statement MAX: New operand MOVE-BUNDLE-MSGs= for controlling automatic moving.
- Administration interface KDCADMI, data structure *kc\_max\_par\_str*: New field *move\_bundle\_msgs* for output of the generated value.

## Other changes

- Output of the generation time at the administration interface:
  - KDCADMI data structure *kc\_curr\_par\_str*: New fields *gen\_date\_xxx* and *gen\_time\_xxx* for the output of the date and time of generation.
  - Command KDCINF STAT: Additional output of the date and time of generation.
- Message when a message to OSI TP and LU6.1 partner is lost

If a message to an OSI TP and LU6.1 partner is deleted because it could not be delivered due to a permanent error, the new message K239 is output.
- Improved performance during communication between UPIC clients and UTM server applications.
- Modification of UTM messages on Unix, Linux and Windows systems
  - Message texts of K and P messages can also be changed on Windows.  
To do this, the UTM tool KDCMMOD and the new UTM tool KDCWMSG are used.
  - Message texts and targets of U messages can be changed on Unix, Linux and Windows systems. As a result, U messages can also be output to the system log file SYSLOG.  
  
The existing procedure for changing K and P messages can now also be used for U messages.
- The port number range for Unix, Linux and Windows systems is no longer restricted, i.e., all port numbers from 1 to 65535 can be specified.
- For CON statements as well as PTERM and TPOOL statements with PTYPE=APPLI/UPIC-R/SOCKET, the specification of the PRONAM= parameter is now mandatory for Unix, Linux and Windows systems. The same applies to the NETWORK-SELECTOR parameter of the OSI-CON statement.

- The UTM tool KDCEVAL has been extended as follows:
  - New operand SHOW-TSN in the KDCEVAL command OPTION for activating or deactivating the repeated output of the TSN or the PID in the TRACE2 list.
  - Additional outputs in the list header, such as appli mode or test mode.
- Encryption
  - BS2000 systems: The encryption functionality of the BS2000 system is used.
  - Unix, Linux and Windows systems: The functionality of the OpenSSL library is used.

### 1.3.2 Discontinued server functions

Several functions have been deleted and can no longer be generated in KDCDEF. If they are still specified, this will be rejected with a syntax error in the KDCDEF run.

The following has been deleted:

- *BS2000 systems*
  - KDCDEF statement USER: Operand CERTIFICATE= and CERTIFICATE-AUTHORITY= for assigning certificates.
  - KDCDEF statement DATABASE: Operand value TYPE=PRISMA for the database system PRISMA.
- *Unix, Linux and Windows systems*
  - The conversion file between short and long computer names (mapping function) and the environment variable UTM\_NET\_HOSTNAME are no longer supported, because by default openUTM supports computer names of up to 64 characters in length.
  - KDCDEF statement MAX: Operand NET-ACCESS= for setting up the network connection (single-threaded/multi-threaded). The multi-threaded network connection is always used.
  - KDCDEF statement FORMSYS for a formatting system.
  - KDCDEF statement OPTION: Operand CHECKTNS= for comparing the UTM generation with the TNS generation.

### 1.3.3 New client functions

#### UPIC routing

A list of partner applications can be specified in the program or via static configuration in the `upicfile`. This enables the implementation of, e.g., flexible addressing or load distribution. Up to now, this type of functionality was only offered for the static configuration of a UTM cluster application.

The following interfaces have been changed to support this:

- `upicfile`: New code ND that can be used to define a list of standalone UTM applications.
- CPI-C interface:
  - New call *Set\_Partner\_Index* for setting the index for a list entry of a partner application. The partner application must be defined then using *Set\_Partner\_xyz* calls. Further partner applications can be inserted subsequently in the partner list.
  - New call *Get\_Max\_Partner\_Index* for querying the maximum index of the list of partner applications.

#### Other changes

- 64-bit support of openUTM client for the AIX and HP Itanium platforms.
- Support of long passwords for UTM user IDs.

The following interfaces have been changed to support this:

- CPI-C calls *Set\_Conversation\_Security\_Password* and *Set\_Conversation\_Security\_New\_Password*:  
Both the existing password and the new password for the UTM user ID can be up to 16 characters in length.
- XATMI call *tpinit*:  
The password for the UTM user ID can be up to 16 characters in length.
- Code conversion tables

For the code conversion between ASCII-compatible codes and EBCDIC codes on the client side, openUTM client provides tables for four code conversions on all platforms (two 8-bit conversions and two 7-bit conversions). By default, an 8-bit conversion between ISO8859-i and EBCDIC.DF.04-i is used on BS2000, Unix, and Linux systems, and an 8-bit conversion between Windows 1252 and EBCDIC.DF.04-F is used on Windows systems.



The tables can be adjusted. Only one table can be used in a client application at any one time.

- Encryption

- BS2000 systems: The encryption functionality of the BS2000 system is used.

Unix, Linux and Windows systems: The functionality of the OpenSSL library is used.




### **1.3.4 New functions for openUTM WinAdmin**

WinAdmin supports all new features of openUTM V6.5 relating to the program interface for the administration and extension of the DADM call at the KDCS interface.

### **1.3.5 New functions for openUTM WebAdmin**

WebAdmin supports all new features of openUTM V6.5 relating to the program interface for the administration and extension of the DADM call at the KDCS interface.

## 1.4 Notational conventions

- |   |   |
|---|---|
| B<br>B  | This symbol is used in the left-hand margin to indicate BS2000 system specific elements of a description.   |
| X<br>X  | This symbol is used in the left-hand margin to indicate Unix and Linux system specific elements of a description.                                       |
| W<br>W  | This symbol is used in the left-hand margin to indicate Windows specific elements of a description.   |
| B/X<br>B/X  | This symbol is used in the left-hand margin to indicate parts of the description that are only relevant for openUTM on BS2000, Unix and Linux systems.  |
| B/W<br>B/W  | This symbol is used in the left-hand margin to indicate parts of the description that are only relevant for openUTM on BS2000 and Windows systems.      |
| X/W<br>X/W  | This symbol is used in the left-hand margin to indicate parts of the description that are only relevant for openUTM on Unix, Linux and Windows systems. |
|  | Indicates references to comprehensive, detailed information on the relevant topic.  |
|  | Indicates notes that are of particular importance.  |
|  | Indicates warnings.   |
| X/W<br>X/W<br>X/W   | <i>utmpath</i><br>On Unix, Linux and Windows systems, designates the directory under which openUTM was installed.                                       |
| B<br>B  | <i>\$userid</i><br>On BS2000 systems, designates the user ID under which openUTM was installed.   |

---

## 2 Communicating via X/Open interfaces

The continuing growth in the networking of computer systems represented the starting point for the creation of new, heterogeneous application architectures on the basis of the client/server model. The integration of differing hardware platforms and software components is a precondition for the development of such networks.

X/Open has defined a development environment, the "Common Applications Environment (CAE)", which provides standards and interfaces for the implementation of open systems.

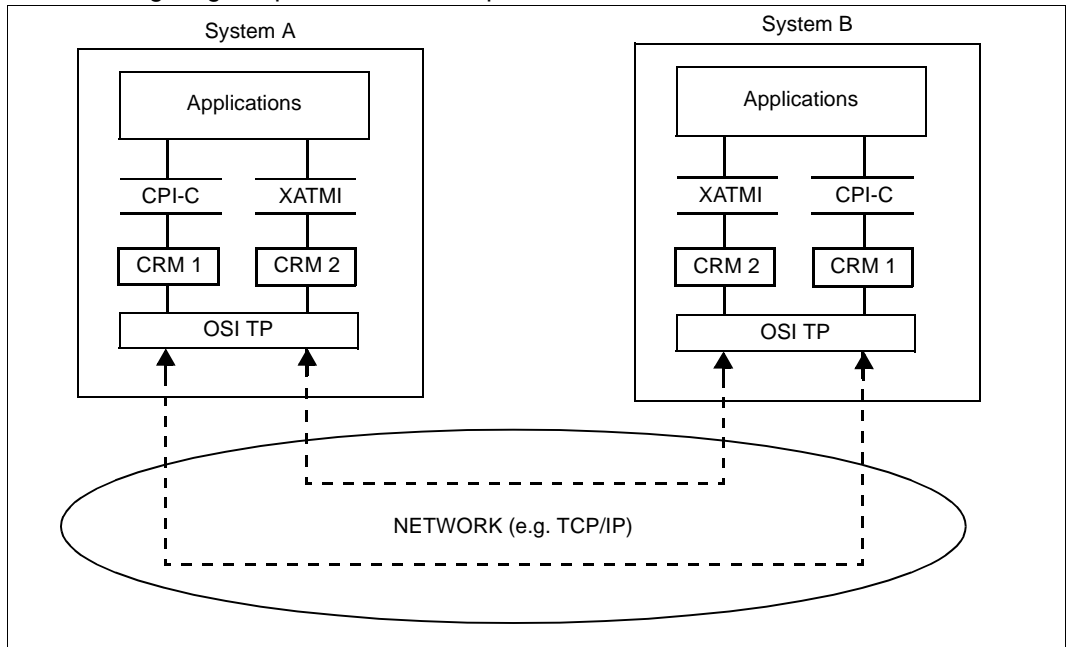
The "Distributed Transaction Processing" (DTP) reference model which was developed by X/Open as part of the "Common Applications Environment" forms the basis for open systems in the field of transaction processing.

The aspects of the reference model which are relevant to openUTM are presented in the next section. For a complete description of the DTP reference model, refer to the X/Open User Guide "Distributed Transaction Processing: Reference Model Version 2".

openUTM not only supports the interfaces which are defined within the framework of "Distributed Transaction Processing" but also provides an optimized range of possibilities for the integration of existing software components such as IBM or BS2000 mainframe services. For an overview of openUTM connectivity, refer to the openUTM manual "Concepts und Functions".

## 2.1 The “Distributed Transaction Processing” reference model

The following diagram presents an excerpt from the DTP reference model



The Transaction Manager (TM) controls global transactions in accordance with the “Distributed Transaction Processing” model. The Transaction Manager is responsible for starting transactions and coordinating commit and rollback decisions as well as for restarts following malfunctions or system crashes.

Communications via two transaction applications are performed via Communication Resource Managers (CRM). Communication Resource Managers communicate via the OSI TP protocol (Open Systems Interconnection - Distributed Transaction Processing) which has been defined by the ISO.

OSI TP may be based on a variety of network protocols such as TCP/IP.

The OSI TP Communication Resource Manager permits an UTM application to communicate with other applications which also use the OSI TP protocol. In particular, it is possible to connect an UTM client application which is running under a Unix, Linux or Windows system with an OpenCPIC or UPIC carrier system. For example, this allows you to provide users with a graphical user interface.

Moreover, in openUTM the OSI TP CRM allows you to connect to applications which use the LU 6.2 communication protocol via an LU62 gateway.

In addition to the communications via OSI TP which are requested by X/Open, CPI-C programs and XATMI programs in openUTM are able to communicate via the protocols LU6.1 and UPIC.

*Interface between application program and Transaction Manager*

X/Open has defined the TX interface between the application program and the Transaction Manager, see „The TX (Transaction Demarcation) Specification“.

*Interface between application program and Communication Resource Manager*

X/Open has defined the CPI-C and XATMI interfaces between the application program and the CRM, see „The CPI-C Specification, Version 2“ und „The XATMI-Specification“.

Please refer to the section [“Related publications” on page 203](#) for the exact titles of the X/Open specifications.

**Description of X/Open interfaces in this User Guide**

This User Guide describes how you can use X/Open interfaces in openUTM. The calls themselves are not described.

Consequently it is essential that you are familiar with the relevant X/Open specifications before you can create services in openUTM which use CPI-C, XATMI or TX calls.

## 2.2 Integrating the X/Open interfaces

In addition to the open interfaces, openUTM uses the KDCS interface to support a national standard which provides all the calls necessary for programming a transaction application as part of a global interface.

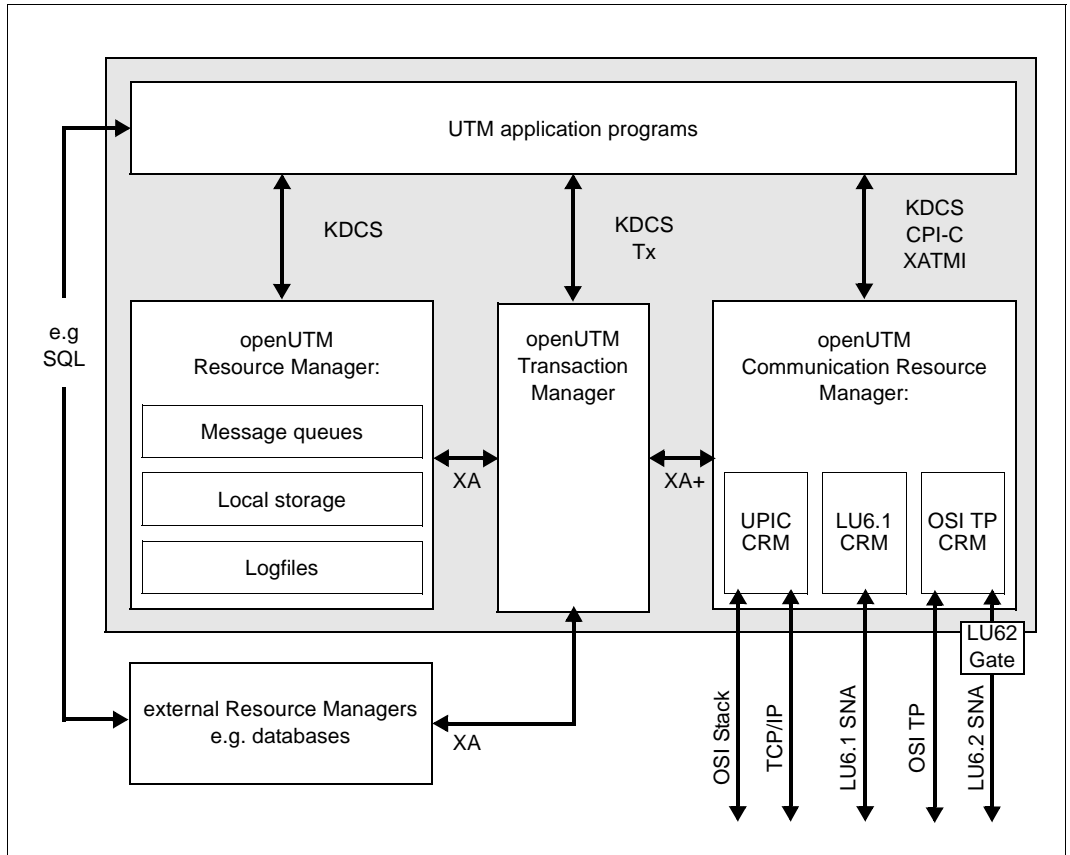
*KDCS program interface*

The KDCS interface provides extensive possibilities for transaction control and program-to-program communications. In addition, KDCS provides transaction-oriented message queueing which permits the time-driven transmission of messages.

You can also use KDCS calls for the transaction-oriented access to openUTM resources such as storage areas and logfiles.

The KDCS interface is described in the *openUTM* manual "Programming Applications with KDCS".

The figure below depicts the interfaces which are provided by openUTM:



Since openUTM can be used on all the usual hardware platforms and because it provides excellent connectivity, you can distribute your application within a heterogeneous environment in a way which responds to the needs of your company's working routines. You can integrate existing application program units even if they run under different transaction monitors. This capability protects your existing investments.

Thus the openUTM open, universal transaction monitor allows you to define the application architecture which is best suited to your business procedures within a heterogeneous IT world and implement this architecture with the appropriate interfaces.

### 2.3 CPI-C , XATMI and TX under openUTM

The interfaces CPI-C, XATMI and TX in openUTM meet X/Open’s CAE specifications for CPI-C V2.0, XATMI and TX.

#### CPI-C and XATMI

are program interfaces for program-to-program communications across hosts. They are available in both COBOL and C.

Under openUTM, CPI-C and XATMI can communicate using the protocols LU6.1 and UPIC in addition to OSI TP.

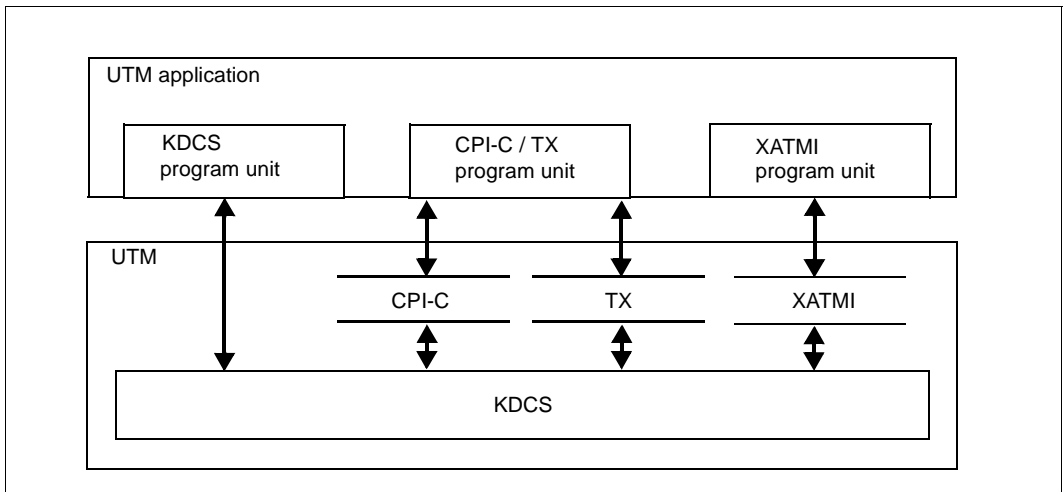
CPI-C can be used for both synchronous and asynchronous communications in Conversational Mode while XATMI can be used for both synchronous and asynchronous communications Request/Response Mode and Conversational Mode.

#### TX

is a program interface used to define transaction boundaries across hosts. They are available in both COBOL and C.

Under openUTM, TX calls are only of relevance in CPI-C program units. In XATMI program units, you implicitly define the transaction boundaries using the XATMI call tpreturn() (see [page 141](#)).

The figure below shows how the X/Open programming interfaces are integrated into openUTM .

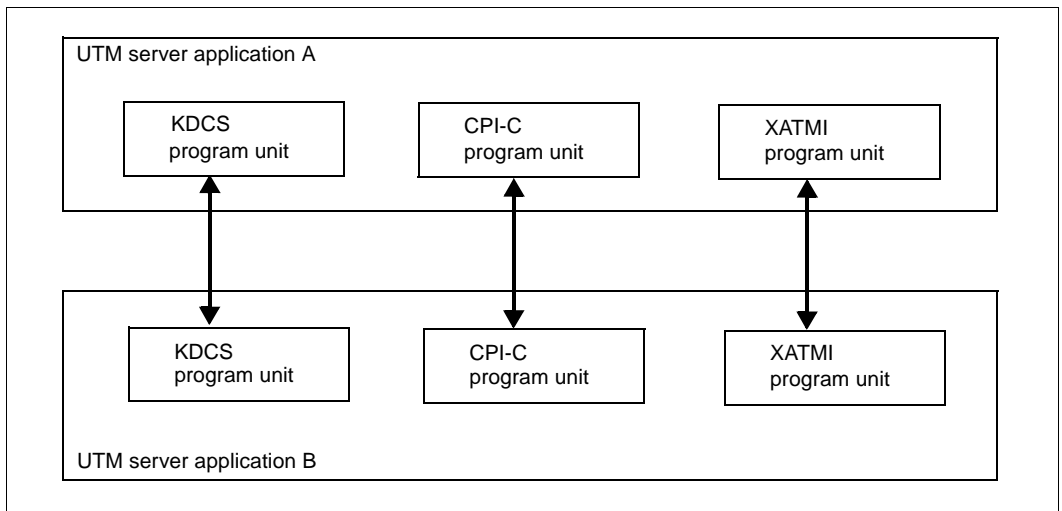


An application running in openUTM may contain program units that use the functions of the program interfaces KDCS, CPI-C or XATMI. Only the calls of one of these interfaces can be used within a program unit. CPI-C program units may also contain TX calls for transaction control. Database calls, for example, can of course be used for processing within a program unit.

During communication between UTM program units within a service, you may use either KDCS calls only or CPI-C calls only or use XATMI calls only, i.e. you may use only one of these interfaces within a service.

## 2.4 Linking server/server with CPI-C and XATMI under openUTM

During communications between UTM server applications, the services (program units) which are in communication with one another must use the same interface.

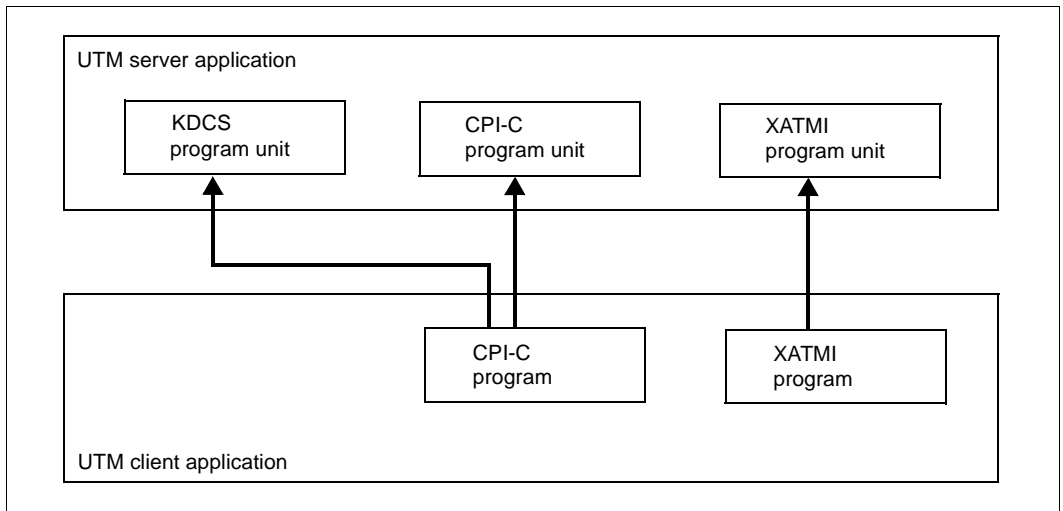




## 2.5 Linking client/server with CPI-C and XATMI under openUTM

The following communication options are available when linking the client/server, i.e. when communicating between the UTM client application and the UTM server applications:

- CPI-C client programs can communicate both with KDCS program units and with CPI-C services (program units) of the UTM server application.
- XATMI client programs can only communicate with XATMI services (program units) of the UTM server application.



Client/server link

The following products from the openUTM product family are available for linking the client/server with CPI-C and XATMI in openUTM:

- For server applications:
  - openUTM (BS2000 systems)
  - openUTM Enterprise Edition (Unix, Linux and Windows systems)

You use these products to create server applications on hosts running BS2000, Unix, Linux or Windows systems.

- For client applications:
  - openUTM-Client (BS2000 systems)
  - openUTM-Client (Unix and Linux systems)
  - openUTM-Client (Windows systems)

You use these products to create client applications on hosts running BS2000, Unix, Linux or Windows systems.

## 2.6 Fields of application for CPI-C, XATMI, TX and KDCS

The following table indicates the main areas where these interfaces can be used:

Interface	Fields of application
KDCS	Is a complete transaction monitor interface which, in addition to functions for program-to-program communication, contains further important transaction functions such as functions for communicating with terminals, for creating asynchronous jobs (background jobs and output jobs), etc. KDCS is suitable for communication between UTM applications and IMS or CICS via LU6.1.
CPI-C	Is an interface for program-to-program communication. It is suitable for the connection of presentation clients (PC, workstation) to UTM applications and for communication between server applications. Because CPI-C is also defined within the framework of IBM SAA, CPI-C is particularly suitable for applications that are to be implemented in the IBM environment.
XATMI	Is an interface for program-to-program communication. Since XATMI is also supported by other transaction monitors, XATMI is particularly suitable for applications which are to communicate with such monitors, e.g. TUXDO.
TX	Is an interface between program and transaction monitor and is used for controlling global transactions. It is particularly suitable for combining OpenCIC applications and UTM applications.

## 2.7 Files and libraries for the X/Open interfaces

Include files for C and COPY elements for COBOL are supplied to help you create UTM server programs which use the X/Open interfaces.

The UTM X/Open library must be linked when you link these programs.

The tool `xatmiggen` is supplied for XATMI programs. This constructs the Local Configuration File (LCF) and supports you during KDCDEF generation.

### X/W Unix, Linux and Windows systems

X/W On Unix, Linux and Windows systems you will find the files and libraries at the following locations, where *interface* means the associated X/Open interface (`cpic`, `tx` or `xatmi`):

X/W ● The include files in the directory


X *utmpath/interface/include* (Unix and Linux systems)  
W *utmpath\interface\include* (Windows systems)

X/W ● The COPY elements in the directory

X *utmpath/interface/copy-cobol85* bzw. *utmpath/interface/netcobol* (Unix and Linux systems)  
X *utmpath\interface\copy-cobol85* bzw. *utmpath\interface\netcobol* (Windows systems)  
W

X ● The X/Open library on Unix and Linux systems is called

X *utmpath/sys/libxopen*

W  On Windows systems, the X/Open functions are included in the UTM library *libwork.lib*.  
W

X/W ● The tool `xatmiggen` is located in directory

X *utmpath/xatmi/ex* (Unix and Linux systems)  
W *utmpath\xatmi\ex* (Windows systems)

### B BS2000 systems

B On BS2000 systems, the include files and COPY elements take the form of type S library elements in the X/Open library

B *\$userid.SYSLIB.UTM.065.XOPEN*

B The tool `xatmiggen` can be found in the following library as a type L library element:

B *\$userid.SYSLNK.UTM.065.UTIL*



---

## 3 X/Open CPI-C interface

This chapter describes the special features of CPI-C under openUTM. This description must be viewed as an openUTM-specific supplement to “Distributed Transaction Processing: The CPI-C Specification, Version 2”. Knowledge of the X/Open specification is thus essential for understanding the concepts and for creating CPI-C programs under openUTM.

### 3.1 The X/Open interface CPI-C

CPI-C (**C**ommon **P**rogramming Interface for **C**ommunication) is a program interface for program-to-program communication beyond system boundaries which has been standardized by X/Open and the CIW (**C**PI-C Implementor's **W**orkshop).

Since CPI-C only supports program-to-program communication it does not offer any functions for communication with terminals. This is the reason why you cannot start CPI-C program units in openUTM directly from a terminal (by entering a transaction code).

Service requests can arrive from UTM clients applications or from other server applications.

In order to process these requests CPI-C program units may request openUTM services from other UTM applications or from remote applications. In this case we speak of server/server network where the requesting partner assumes the role of the client.

### 3.1.1 Definition of terms

#### *Conversation*

In CPI-C, the communication between two CPI-C application programs is known as a *conversation*. In openUTM/KDCS terminology this is known as a *UTM service*.

A conversation is established on an existing logical connection between the partner applications containing the CPI-C programs. Logical connections are called *associations* in conjunction with OSI TP, *sessions* in conjunction with LU6.1 and *conversations* in conjunction with UPIC clients.

Both partners of a conversation assume different roles within the conversation. One partner initiates the conversation; this partner is referred to below as the *initiator of the conversation*. The other partner accepts the conversation connect and is called the *acceptor*. As far as the roles of client and server are concerned, the client is the initiator and the server is the acceptor.

#### *Initiator of a conversation/outgoing conversation*

The *initiator* of a conversation actively sets up the conversation by using the CPI-C calls Initialize\_Conversation and Allocate. It initializes the conversation. In the openUTM/KDCS terminology, the term *initiator* of a conversation is equivalent to the term *job submitter* in the case of server/server communication and to the term *client* in the case of client/server communication.

A conversation in which the local CPI-C program is the initiator is called an *outgoing conversation*. In the X/Open specification, the terms *outbound conversation* and *outgoing conversation* are used synonymously.

#### *Acceptor of a conversation/incoming conversation*

The *acceptor* of a conversation accepts the conversation by using the CPI-C call Accept\_Conversation. In the openUTM/KDCS terminology, the term *acceptor* of a conversation is equivalent to the term *job receiver* in the case of server/server communication and to the term *server* in the case of client/server communication.

A conversation in which the local CPI-C program is the receiver is called an *incoming conversation*. In the X/Open specification, the terms *inbound conversation* and *incoming conversation* are used synonymously.

A CPI-C program can maintain precisely one *incoming* and several *outgoing conversations* under openUTM at any one time, i.e. within a program run the CPI-C program under openUTM can be the acceptor in exactly one conversation and can be the initiator in a number of conversations. The CPI-C program is started under openUTM when the *incoming conversation* request is received.

### *Dialog and asynchronous conversation*

In openUTM, a distinction is made between a dialog conversation and an asynchronous conversation depending on whether the acceptor is an asynchronous service or not.

In *dialog conversations*, both sides of the conversation can send data, i.e. both the initiator (client) or the acceptor (server). A dialog transaction code (specify TYPE=D in KDCDEF-control statement TAC or LTAC) must be defined in the UTM application for the acceptor of a dialog conversation.

In *asynchronous conversations*, only the initiator of the conversation can send data. An asynchronous transaction code (specify TYPE=D in KDCDEF control statement TAC or LTAC) must be defined in the UTM application for the acceptor of an asynchronous conversation.

### *Conversation ID*

An unambiguous conversation ID is assigned locally by CPI-C to each conversation, i.e. the initiator and the acceptor each has its own, local conversation ID which does not have to match that of the ID of the partner. With the conversation ID, each CPI-C call within a program is uniquely assigned to a conversation.

### *Conversation characteristics*

Each conversation has a range of *conversation characteristics*. The values of the characteristics affect the operational sequences of the conversation and the CPI-C functions available in the conversation. Some of the characteristics only determine the operational sequences on one side of the conversation, while others apply to both sides. When the conversation is initialized, the characteristics are preassigned default values. The initialization values of the characteristics depend on the role of the CPI-C program at the start of the conversation, i.e. whether the program initiates or accepts the conversation. The values of the characteristics are modified through the execution of CPI-C calls within the conversation.

The current values of the characteristics can be modified explicitly using Set calls and queried using Extract calls

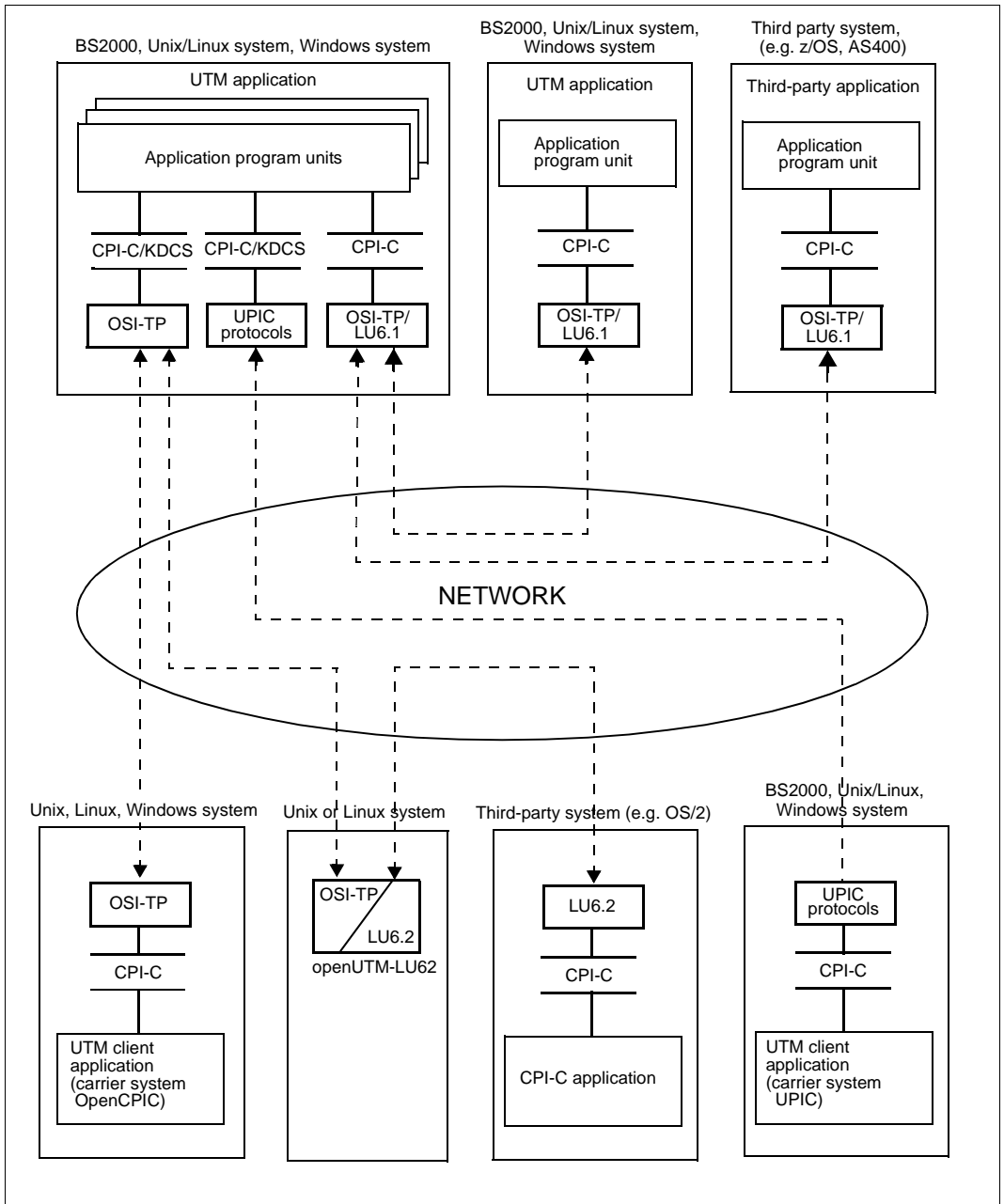
The conversation characteristics that are significant for CPI-C programs under openUTM are listed on page 54ff.

### 3.1.2 Communication partners of a CPI-C application under openUTM

The following diagram indicates the possible communication partners of a CPI-C application under openUTM. These are:

- CPI-C programs in other UTM applications on BS2000, Unix, Linux or Windows systems systems. In this case, the local CPI-C program can act as either the initiator or the acceptor of the conversation.
- CPI-C programs in applications on non-SNI systems that support OSI TP or LU6.1. Here too, the local CPI-C program can act as either the initiator or the acceptor of the conversation.
- UTM client application with carrier system OpenCPIC.  
The local CPI-C program can act as initiator or acceptor of the conversation in UTM client applications with carrier system OpenCPIC.
- UTM client application with the UPIC carrier system.  
An UTM application can only hold incoming conversations with UTM client applications with the UPIC carrier system, i.e. conversations initiated by the UTM client application. The local CPI-C program is always the acceptor in conversations using the UPIC protocol.
- LU6.2 applications in third party systems.  
UTM applications can, for example, hold conversations with CPI-C applications in OS/2 systems that use the LU6.2 protocol. In this case, the connection between the UTM application and the non-SNI system must be established via a Unix, Linux or Windows system on which OpenCPIC and a openUTM-LU62 gateway are installed (see following diagram). No OpenCPIC application need be created in the Unix, Linux or Windows system for the connection.





Communication partners of a CPI-C application under openUTM

### 3.1.3 Linking server/server with CPI-C

In order for two CPI-C partners to set up a conversation, a logical connection must be established between the application entities. The CPI-C partner applications set up the conversation on this connection.

CPI-C programs under openUTM can set up conversations on OSI TP associations and LU6.1 sessions. In this way, a CPI-C program can simultaneously hold conversations using the OSI TP protocol and the LU6.1 protocol.

A CPI-C program under openUTM can also accept a conversation (incoming conversation) from UTM-Clients with the UPIC carrier system using the UPIC protocol, and simultaneously hold outgoing conversations with other server applications using the OSI TP and LU6.1 protocol.

As soon as a request for a conversation is received, openUTM either establishes the necessary association or session, or an existing association/session that has been released by another conversation is made available to the partners for their conversation..

#### **KDCDEF-statements for establishing an association or session**

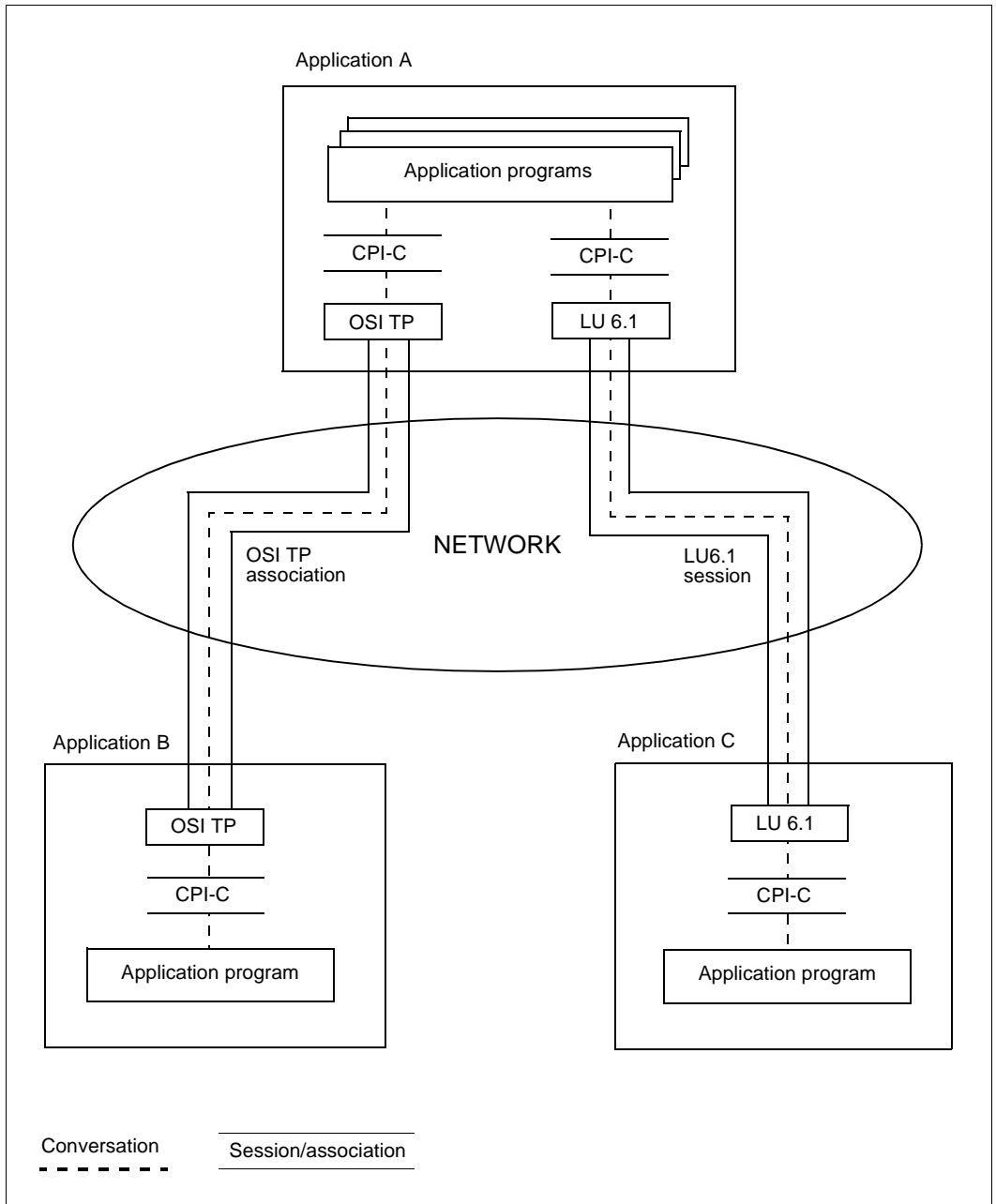
A KDCDEF generation must be present before a UTM application can establish an association or session to the partner.

For detailed information of KDCDEF generation please refer to the openUTM manual "Generating Applications". In the following only those statements are described which are of relevance when you establish a conversation.

In order for openUTM to establish an association you must define the partner application using an OSI-LPAP statement and the connection using an OSI-CON statement.

To set up an LU6.1 session, you must define the partner with an LPAP statement and the connection with CON, SESCHA, and LSES statements.

For an UTM-Client with the UPIC carrier system, you must specify an LTERM and a PTERM statement in the KDCDEF generation or generate a terminal pool with a TPOOL statement.



Linking applications with CPI-C

### 3.1.4 Sample application – flowchart

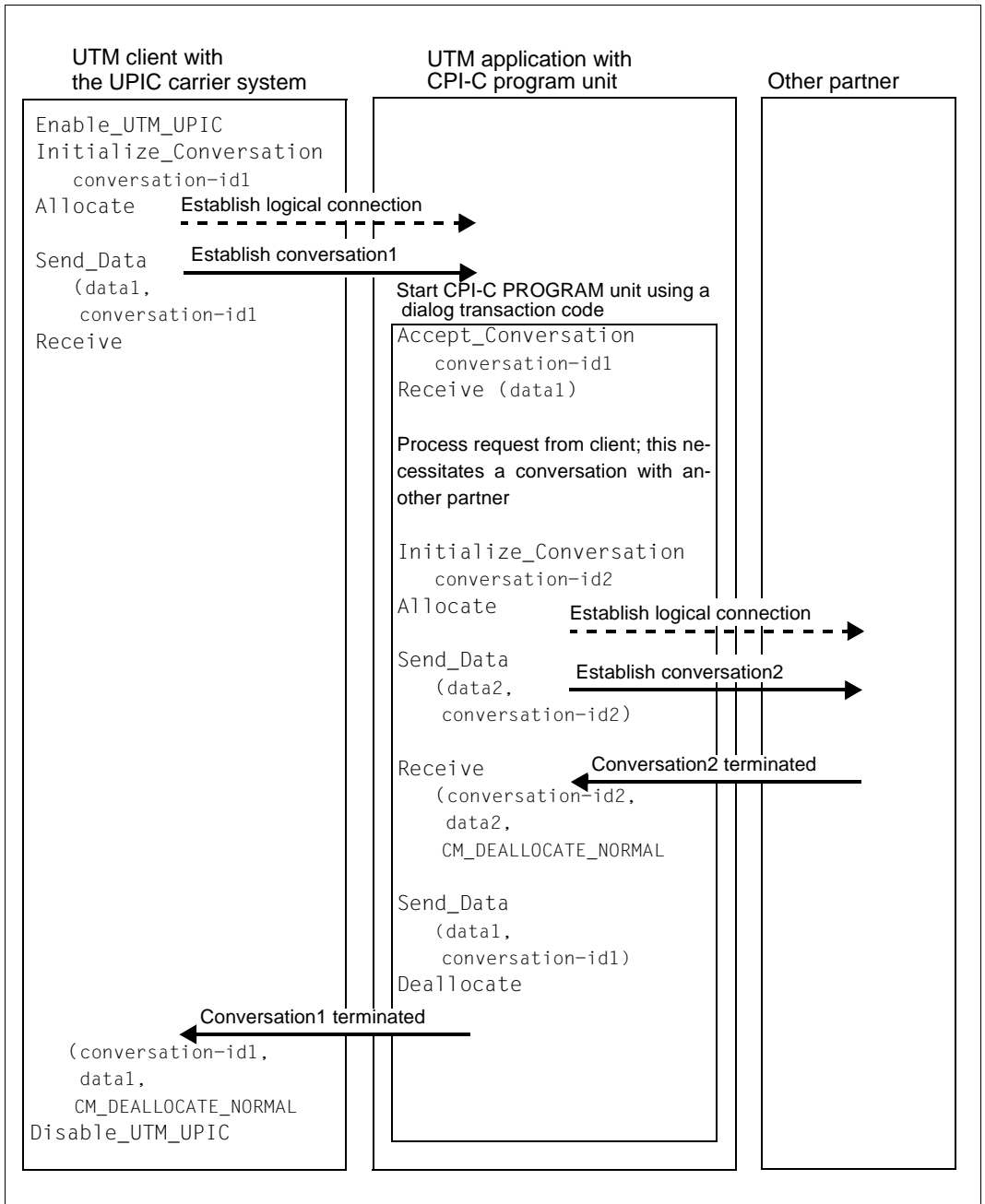
A typical example of linking server/server with CPI-C programs under openUTM is described below.

The CPI-C program unit of an UTM application is started by a request for a conversation (it accepts the conversation). The requesting program which initializes the conversation may, for example, be an UTM-Client, or alternatively a CPI-C program of another application which was itself started by a conversation request and assumes the role of the initiator in the conversation to the next service.

In the following flowchart, the CPI-C program unit under openUTM is started by an UTM-Client with the UPIC carrier system and accepts the conversation (conversation 1). From the point of view of the CPI-C program unit under openUTM, conversation1 is an incoming conversation.

The CPI-C program under openUTM processes the initiator's request and, for example, accesses a database. To process the service request it may be necessary for this CPI-C program itself to initiate a conversation with a CPI-C program in another application (UTM application or application of a non-SNI system), for example because the database that has to be accessed is located on this system (conversation2).

From the point of view of the CPI-C program unit under openUTM, conversation2 is an outgoing conversation.



## 3.2 CPI-C characteristics and functions in openUTM

The aim of this section is to provide an overview of the functions you can use with CPI-C under openUTM. The conversation characteristics, conversation states, and limits relevant to CPI-C under openUTM are also listed.

### 3.2.1 Conversation characteristic `conversation_type`

When communicating via LU6.1 and OSI TP, CPI-C in openUTM supports the conversation type *mapped conversation*. A mapped conversation permits the exchange of data records in any data format which is agreed by both conversation partners. Precisely one data record is transferred in each `Send_Data` call.

The conversation characteristic `conversation_type` can therefore only be assigned the value `CM_MAPPED_CONVERSATION`.

The value `CM_BASIC_CONVERSATION` is rejected with `CM_PRODUCT_SPECIFIC_ERROR`.

### 3.2.2 Conversation characteristics for addressing

Address information is required in order to establish a conversation. In CPI-C, the address information is managed in the side information. The address information managed in the side information can be retrieved from the program using a symbolic name. This name is transferred by the program in the Initialize\_Conversation call in the sym\_dest\_name (symbolic destination name) parameter. The address information is read and the conversation characteristics TP\_name, partner\_LU\_name, AE\_qualifier, AP\_title, and application\_context\_name are supplied with the data from the side information.

In openUTM, the side information is contained in the KDCFILE. The address information is defined in the KDCDEF generation.

#### Incorporating the address information in the KDCFILE

At least one LPAP statement must be issued for each partner application to be communicated with using the LU6.1 protocol. The LPAP statement describes the remote partner application. You must assign CON, SESCHA, and LSES statements to the LPAP statements in the KDCDEF generation. The information defined in these statements is required by openUTM in order to establish the session to the partner application and define the session characteristics.

At least one OSI-LPAP statement must be issued for each partner application to be communicated with using the OSI TP protocol. The OSI-LPAP statement describes the remote partner application. You must assign an OSI-CON statement to the OSI-LPAP statement in the KDCDEF generation. The information defined in the OSI-LPAP and OSI-CON statements is required by openUTM in order to establish the associations to the partner application and define the characteristics of the associations.

An UTM client partner with the UPIC carrier system must be defined with an LTERM statement and a PTERM statement or alternatively, with a TPOOL statement in the KDCDEF generation.

At least one LTAC statement must be issued for each partner application to which a CPI-C program under openUTM wants to establish an outgoing conversation. With the LTAC statement, a transaction code for a service of the remote server application is defined in the local application. The ltac name specified in this LTAC statement is the symbolic name by which the CPI-C program can access the address information.

The LTAC statement together with the LPAP or OSI-LPAP statement contains the information required by the CPI-C program in order to establish a conversation to the partner application.

### *Multiple partner applications*

Multiple partner applications can be grouped together to form a bundle using the statement MASTER-OSI-LPAP or MASTER-LU61-LPAP.

If the program uses a MASTER-OSI-LPAP or MASTER-LU61-LPAP then UTM selects an OSI-LPAP or LPAP from the bundle. For information on which OSI-LPAP or LPAP in a bundle is selected by UTM, see openUTM manual "Generating Applications", section "Generating applications for distributed processing".

### **Transferring address information**

To specify the address information required to establish the conversation, the program transfers to openUTM either only the *ltac* name from the LTAC statement, or the *ltac* name plus the *lpap* name from the LPAP or OSI-LPAP statement or from the MASTER-LU61-LPAP or MASTER-OSI-LPAP statement. The way in which the partner is defined in the KDCDEF generation determines whether only the *ltac* name or both names are transferred.

The programmer has the following options for specifying address information:

- In the KDCDEF generation, the operand LPAP=*lpap-name* (*lpap-name* from the LPAP or the OSI-LPAP statement or from the MASTER-LU61-LPAP or MASTER-OSI-LPAP statement) is specified in the LTAC statement, thereby directly assigning the LTAC an LPAP or OSI-LPAP statement or MASTER-LU61-LPAP or MASTER-OSI-LPAP statement that describes the partner application. In this case an Initialize\_Conversation call is sufficient, whereby the transferred *sym\_dest\_name* has to be identical to the *ltac* name from the LTAC statement. The conversation characteristics *TP\_name*, *partner\_LU\_name*, *AE\_qualifier*, *AP\_title*, and *application\_context\_name* are assigned the values defined in the LTAC and LPAP or OSI-LPAP statements.

If desired, *ltac* and *lpap* name can also be transferred by calling Set\_TP\_Name and Set\_Partner\_LU\_Name. The values of the conversation characteristics transferred in these Set\_ calls override those set with Initialize\_Conversation.

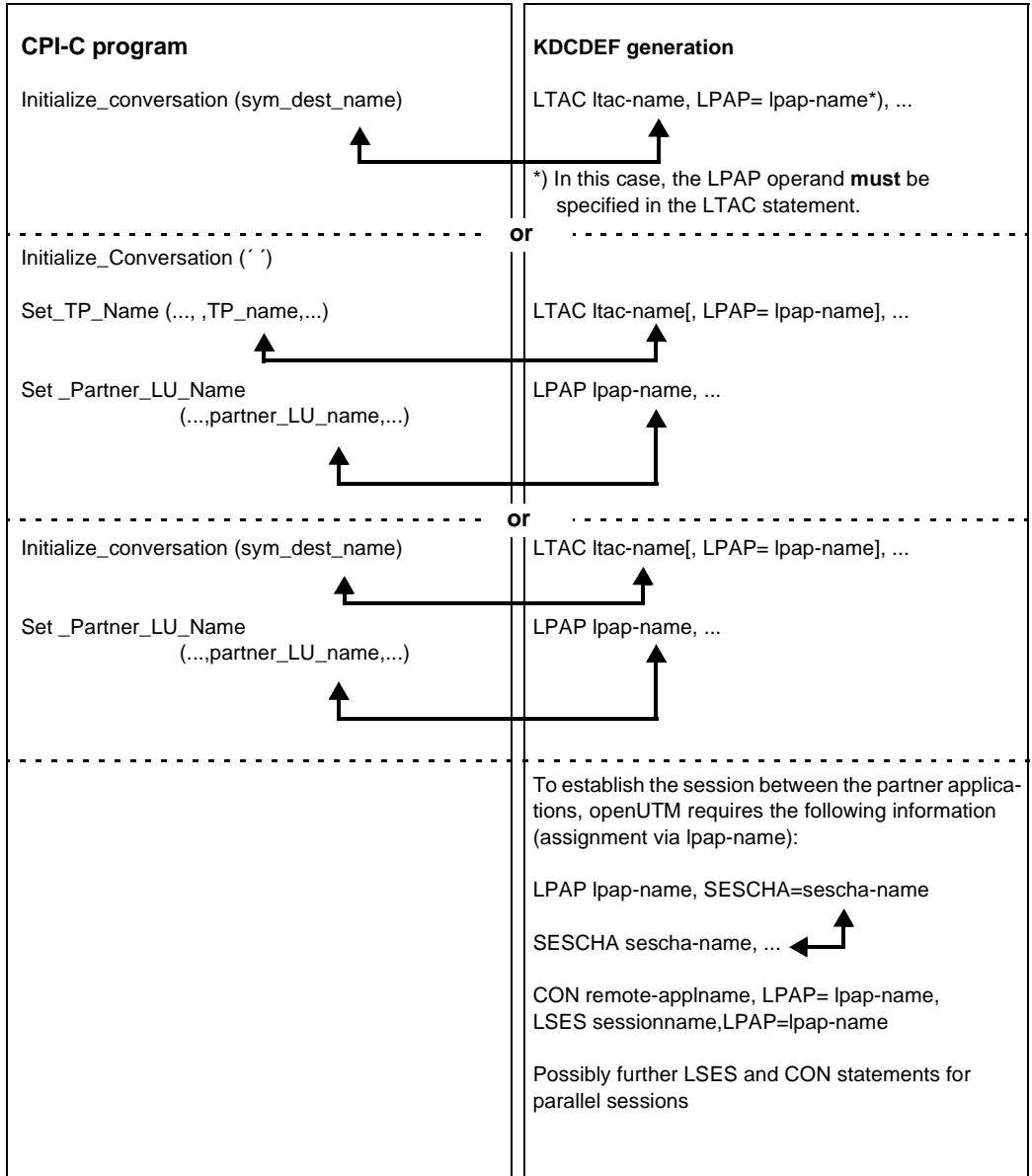
- In the KDCDEF generation, the LPAP operand is not specified in the LTAC statement, i.e. no remote application is assigned to the LTAC in the generation. LTAC must be assigned to LPAP or OSI-LPAP in the CPI-C program. In this case, there are two ways of supplying the address information to the conversation characteristics:
  - You transfer blanks for *sym\_dest\_name* in Initialize\_Conversation and then you invoke the calls Set\_TP\_Name (with *TP\_name=ltac-name*) and Set\_Partner\_LU\_name (with *partner\_LU\_name=lpap-name*) to supply the conversation characteristics before the Allocate call;



- or you specify the *ltac* name from the LTAC statement in the *sym\_dest\_name* parameter when you call `Initialize_Conversation` and you then call `Set_Partner_LU-Name` (with *partner\_LU\_name=lpap-name*) to set `partner_LU_name` in order to specify the missing information on the remote partner application. The call `Set_Partner_LU-Name` must be invoked before the `Allocate` call.

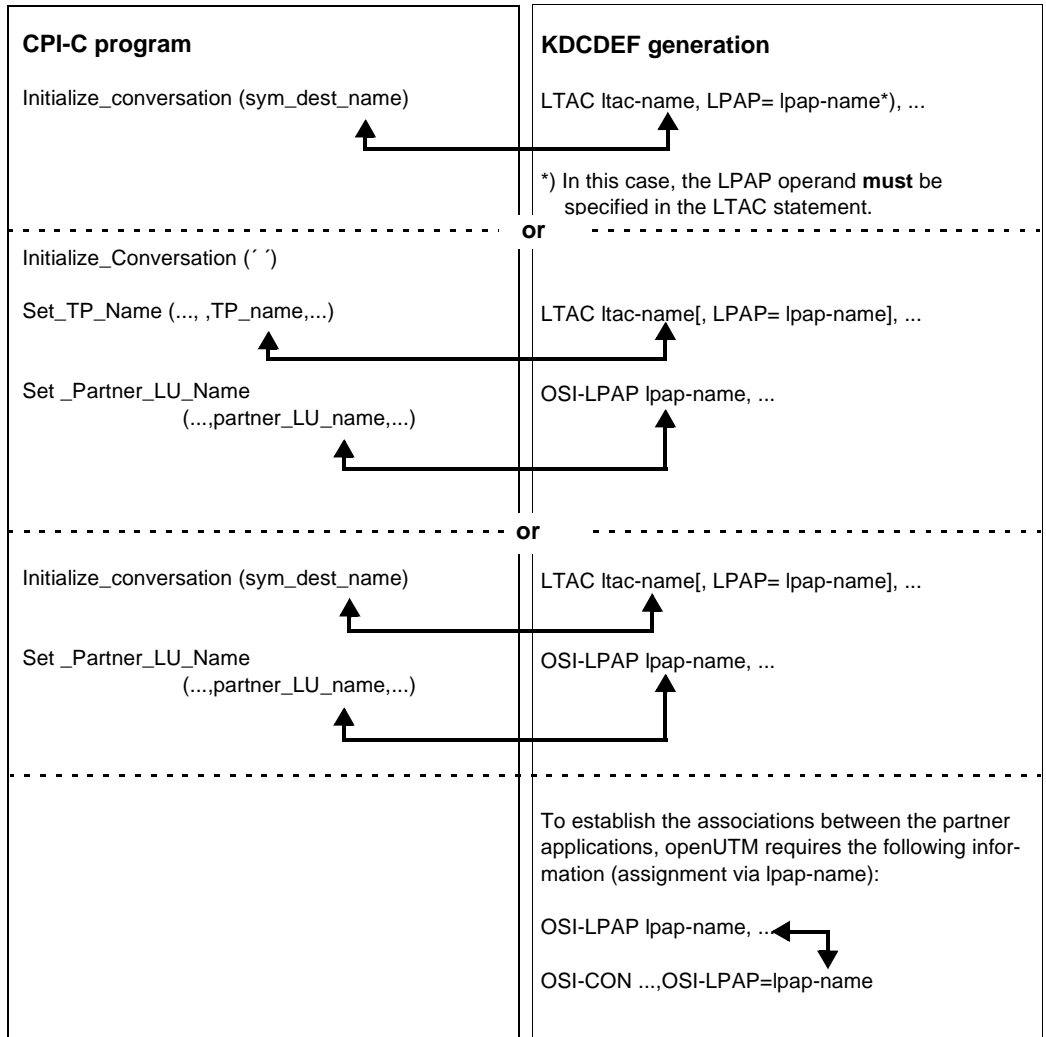
The following diagrams indicate how the information in the CPI-C program and the definitions in the KDCDEF generation must be coordinated for outgoing conversations to OSI TP and LU6.1 partners. The names linked by arrows in the diagrams must match.

*Coordination of CPI-C program calls with KDCDEF generation for conversations via LU6.1 (with individual LPAPs\*)*



\* For information on LPAP bundles, see section [“Multiple partner applications”](#) on page 48.

*Coordination of CPI-C program calls with KDCDEF generation for conversations via OSI-TP with individual OSI-LPAPs \**



\* For information on OSI-LPAP bundles, see section [“Multiple partner applications”](#) on page 48.

**Assignment of conversation characteristics when establishing a conversation**

The following table describes how the conversation characteristics for addressing the partner are assigned when establishing the conversation.

	<b>Characteristic</b>	<b>Value of characteristic with outgoing conversations (initiator)</b>	<b>Value of characteristic with incoming conversations (acceptor)</b>
LU6:1	partner_LU_name	lpap name of the LPAP statement which describes the remote application. The Extract_Partner_LU_Name call only returns a value if partner_LU_name has been set explicitly with the Set_Partner_LU_Name call.	lpap name of the LPAP statement which describes the remote application.
	TP_Name	The ltac name of the LTAC statement.	Name of the transaction code under which the local program was invoked (TAC tac name).
OSI:TP	partner_LU_name	osi-lpap name of the OSI-LPAP statement which describes the remote partner application. The Extract_Partner_LU_Name call only returns a value if partner_LU_name has been set explicitly with the Set_Partner_LU_Name call.	osi-lpap name of the OSI-LPAP statement which describes the remote partner application.
	TP_Name	ltac name of the LTAC statement.	TAC name of the transaction code under which the local program was invoked.
	AE_Qualifier	AEQ from the OSI-LPAP statement which describes the partner application.	AEQ from the OSI-LPAP statement which describes the partner application.
	AP-Title	APT from the OSI-LPAP statement which describes the partner application.	APT from the OSI-LPAP statement which describes the partner application.
	application_context	Value of the operand APPLICATION-CONTEXT of the OSI-LPAP statement.	Value of the operand APPLICATION-CONTEXT of the OSI-LPAP statement.

	Characteristic	Value of characteristic with outgoing conversations (initiator)	Value of characteristic with incoming conversations (acceptor)
UPIC application	partner_LU_name	openUTM cannot establish outgoing conversations to an UTM-Client with the UPIC carrier system.	Name of the LTERM partner via which the connection was established.
	TP_Name		Name of the transaction code under which the local program was invoked (TAC tac name).

### Maximum name length

Unlike X/Open-CPI-C, the maximum length of the name specified for `partner_LU_name` or `TP_name` is 8 bytes for CPI-C under openUTM. If a value greater than 8 bytes is specified in a CPI-C call in the parameter `partner_LU_name_length` or `TP_name_length`, openUTM rejects the call with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

### 3.2.3 Send-receive mode and send control

CPI-C under openUTM supports half-duplex conversations, i.e. only one of the two partners has send control at any one time. The send control can be transferred from the sending partner to the other by setting the `send_type` characteristic to `CM_SEND_AND_PREP_TO_RECEIVE` and then issuing a Receive call, or by calling Receive in *Send* state. The non-blocking call `Prepare_to_Receive` is not supported in openUTM.

Send control can only be transferred from the partner that has send control to the other partner. The `Request_to_Send` call, with which a partner in Receive state can request send control, is not supported by openUTM. If a remote non-UTM partner requests send control by calling `Request_to_Send`, then request is rejected by openUTM and is not forwarded to the local CPI-C program.

Send-receive mode is defined in the `send_receive_mode` characteristic. This must be defined at initialization. The default setting of `send_receive_mode` is `CM_HALF_DUPLEX` and cannot be modified by CPI-C programs under openUTM.

### 3.2.4 Multiple conversations in one CPI-C program

A CPI-C program can simultaneously hold multiple conversations within a program run. Using the various conversation IDs, the CPI-C calls in the program run are assigned to the individual conversations. With the maximum possible number of simultaneous conversations of a CPI-C program, a distinction is made between incoming conversations and outgoing conversations.

#### Incoming conversation

Under openUTM, every new service request to a program unit causes the corresponding program unit to be restarted, i.e. openUTM restarts the program unit as acceptor of the incoming conversation each time a TAC transaction code (tac-name) which is assigned to the program unit is received from a remote partner.

This means that it is not possible for a CPI-C program to process a second incoming conversation.

A CPI-C program can therefore accept a maximum of one incoming conversation with `Accept_Conversation`. A second `Accept_Conversation` call within a program run is answered with `CM_PROGRAM_STATE_CHECK`, which means "No incoming conversation exists".

#### Outgoing conversation

The maximum number of outgoing conversations is limited by the following factors:

1. The maximum number of outgoing conversations that can exist simultaneously in a program run of a CPI-C program..

In addition to the incoming conversation which started the program, six outgoing conversations can exist at the same time.

If an attempt is made to establish a seventh (open) outgoing conversation, the `Initialize_Conversation` call is rejected with `CM_PRODUCT_SPECIFIC_ERROR`

If the CPI-C program is already holding six outgoing conversations and one of these is terminated, the program can establish a new outgoing conversation.

2. The maximum number of outgoing conversations that the local application can hold simultaneously with a particular partner application.

This number is limited by the maximum permitted number of parallel OSI TP associations or LU6.1 sessions that can exist simultaneously to a partner application.

The maximum number of parallel associations to an OSI TP partner is defined in the ASSOCIATIONS operand in the OSI-LPAP statement. The maximum number of parallel sessions to an LU6.1 partner is limited by the number of LSES statements assigned to the LPAP statement of the partner.

3. The maximum number of outgoing conversations that the local application can hold simultaneously.

This number is limited by the maximum total number of associations and sessions that the local UTM application can hold simultaneously.

This value is determined by the product derived from the sum of all generated OSI TP associations and LU6.1 sessions (number of LSES statements) multiplied by the value of MAXJR (specified in ).

$(\text{number of associations} + \text{number of sessions}) * \text{MAXJR}$

The value of MAXJR is defined in the UTMD statement in the KDCDEF generation.

If one of these values is exceeded by a requested outgoing conversation, the respective Allocate call is rejected with the return code CM\_ALLOCATE\_FAILURE\_NO\_RETRY (for the list of possible return codes see the comments on the Allocate call on [page 71](#)).

### 3.2.5 Conversation characteristic `sync_level`

The level of synchronization when processing between the two CPI-C programs of a conversation is defined in the conversation characteristic `sync_level`. The value of `sync_level` must be set before the conversation is established.

CPI-C programs under openUTM can set the value `CM_NONE` or `CM_SYNC_POINT_NO_CONFIRM` for `sync_level`.

`CM_NONE` means that no synchronization is requested for the conversion - neither synchronization by confirmation (request for confirmation) nor Sync Pointing (global transaction management).

`CM_SYNC_POINT_NO_CONFIRM` means that the conversation is included in a global transaction.

If a CPI-C program under openUTM attempts to set a value other than `CM_NONE` or `CM_SYNC_POINT_NO_CONFIRM` by calling `Set_Sync_Level`, openUTM rejects the call as follows:

- `sync_level=CM_CONFIRM`  
with the return code `CM_PRODUCT_SPECIFIC_ERROR`
- `sync_level= CM_SYNC_POINT`  
with the return code `CM_PARM_VALUE_NOT_SUPPORTED`.

#### Conversation with a non-UTM application

If a CPI-C program under openUTM holds a conversation with a CPI-C program of a non-UTM application, the partner program may request synchronization of the conversation with `sync_level` `CM_CONFIRM` or `CM_SYNC_POINT`.

The program running under openUTM must use `Extract_Sync_Level` to request the set value and react accordingly.

- The non-UTM program has set `CM_CONFIRM` and requests a confirmation (receive confirmation), e.g. by calling `Set_Prepare_To_Receive_Type` with `prepare_to_receive_type = CM_PREP_TO_RECEIVE_CONFIRM` and then calling `Prepare_To_Receive`.

In CPI-C under openUTM, the calls `Confirmed` and `Send_Error` are available and can be used by the CPI-C program under openUTM to send a positive or negative confirmation to the partner program.

- The non-UTM program has set `CM_SYNC_POINT` for `sync_level`.

The CPI-C program under openUTM may not terminate the conversation itself but has to wait until the client requests end of transaction and terminates the conversation (`CM_TAKE_COMMIT_DEALLOCATE` or `CM_TAKE_BACKOUT` in `status_received`).



### 3.2.6 Maximum message length

The length of messages is limited by the maximum buffer size `CM_MAXIMUM_BUFFER_SIZE`. In openUTM this is set to 32767 bytes.

The maximum buffer size is relevant for the parameters `buffer_length`, `send_length`, and `requested_length`. If a value outside the range  $0 \leq \dots\_length \leq CM\_MAXIMUM\_BUFFER\_SIZE$  is specified for one of these parameters, the call is rejected with `CM_PROGRAM_PARAMETER_CHECK`.

### 3.2.7 Converting characteristics and user data

#### 3.2.7.1 Converting characteristics

Some of the conversation characteristics do not define the conversation only locally. These characteristics must be transferred to the conversation partner. They must arrive at the partner encoded in a format that is “understood” by this partner. The characteristics must be converted in accordance with the character sets used on the partner system. The conversion is performed automatically and is not the programmer’s concern.

When communicating via LU6.1, the characteristic data is converted from the locally used system code into EBCDIC, if necessary. If CPI-C receives characteristic data from the partner, then CPI-C assumes that the data is in EBCDIC format and converts it into the character set used on the local system.

When communicating via OSI TP, the characteristic data is converted into the transfer syntax that was defined when generating the UTM application for associations between the partner applications (OSI-LPAP statement, operand `APPLICATION-CONTEXT`). The characteristic data is converted automatically into this transfer syntax.

#### 3.2.7.2 Converting user data

For converting user data, the CPI-C interface provides the programmer with the calls `Convert_Outgoing` and `Convert_Incoming`.

X/W On Unix, Linux and Windows systems, UTM alternatively offers the possibility to generate  
 X/W the conversion of the exchanged user data via KDCDEF. To do this specify one of the  
 X/W values `SYSTEM`, `SYS1`, `SYS2`, `SYS3` or `SYS4` in the `MAP=` operand of the `OSI-CON` or  
 X/W `SESCHA` statement. Please consider the note on [page 59](#) in this case.

#### **Convert\_Outgoing (CMCNVO)**

You can use `Convert_Outgoing` to convert data before sending.

B B B	CPI-C on BS2000 systems assumes that the data transferred in the “buffer” parameter is encoded in EBCDIC.DF.04.1 character code and converts it into an “EBCDIC Multilingual 697/1 Code Page 500/1” character set.
X/W X/W X/W	CPI-C on Unix, Linux or Windows systems assumes that the data transferred in the “buffer” parameter is encoded in ISO 8-bit code ISO 8859-1 (ASCII) and converts it into a modified “EBCDIC Multilingual 697/1 Code Page 500/1” character set.
X/W X/W X/W X/W X/W X/W X/W	The EBCDIC Multilingual 697/1 Code Page 500/1 character set (referred to below as simply EBCDIC Code Page 500/1) is described in “X/Open CAE Specification CPI-C”, Appendix A “Character Sets“ . n This character set was modified for openUTM on Unix, Linux and Windows systems in such a way that the new-line character (n) functions in the same way on Unix, Linux or Windows systems and BS2000 systems following conversion. This was achieved by swapping the location of the characters 0x15 and 0x25 in the code table.

### Convert\_Incoming (CMCNVI)

The data received can be converted with the Convert\_Incoming call. CPI-C assumes that the data is encoded in the character set EBCDIC Code Page 500/1.

B	On BS2000 systems this call converts the data into the EBCDIC.DF.04.1 character set.
X/W X/W	On Unix, Linux or Windows systems this call converts the data into the ISO 8859-1 (ASCII) character set.

### Homogenous link

With a homogeneous link, the partner applications are located both on BS2000 systems or both on Unix, Linux or Windows systems.

Depending on the protocol used (OSI-TP or UPIC), the following applies:

- Server/server communication via OSI-TP
 

With a homogeneous link between two UTM applications (CPI-C applications) or between an Open-CPIC-Client (CPI-C Client with OpenCPIC carrier system) and a UTM application (CPI-C application), conversion is permitted but is not necessary. However, both partners must do the same thing, i.e. both convert or neither converts. If, for example, one side converts and the other does not in a link between two Unix, Linux or Windows partners, the result may be an indecipherable code.
- Client/server communication via UPIC
 

With a homogeneous link between a UPIC client (CPI-C client with UPIC carrier system) and a UTM application (CPI-C application) the conversion functions **must not** be used.

## Heterogeneous link

With a heterogeneous link, one partner application is located on a BS2000 system and the other partner application is located Unix, Linux or Windows system.

Depending on the protocol used (OSI-TP or UPIC), the following applies:

- Server/server communication via OSI-TP

With a heterogeneous link, the sender should encode the data with the `Convert_Outgoing` call and the receiver should do the same with the `Convert_Incoming` call. Please consider the notes below.

- Client/server communication via UPIC

With a heterogeneous link, the UPIC client should perform the code conversion, regardless whether the UPIC client is running on the BS2000 system or on the Unix, Linux or Windows system. Please consider the notes below.

### Notes



The EBCDIC variants EBCDIC.DF.04.1 and EBCDIC Code Page 500/1 differ in 17 characters (e.g. in [ ] { } \ | !, see code conversion table on [page 60](#)). If these characters do not exist in the user data then conversion is not necessary, e.g., for communication between a BS2000 and a Unix or Linux system.



### Caution!

If the calls `Convert_Incoming` and `Convert_Outgoing` are used for data conversion, no automatic conversion can be generated by openUTM, i.e. `MAP=USER` **must** be specified in the OSI-CON statement (with OSI TP partners) or in the SESCHA statement (with LU6.1 partners) when generating the connections to the partner application.

When you use the conversion calls please make sure that the calls are used in the correct sequence since data which is converted repeatedly with `Convert_Incoming` or `Convert_Outgoing` can no longer be interpreted in the target system. The CPI-C trace will indicate whether `Convert_Incoming` and `Convert_Outgoing` are correctly coordinated at both partners.

**Code conversion tables**

This section contains the code conversion tables used by CPI-C in the calls `Convert_Incoming` and `Convert_Outgoing`. The conversion tables shown are those used in `openUTM` on BS2000 systems and `openUTM` on Unix, Linux and Windows systems.

**B Conversion tables for openUTM on BS2000 systems**

**B** The following table lists only those 17 characters that have different codes in  
**B** EBCDIC.DF.04.1 and EBCDIC Code Page 500/1. `Convert_Incoming` and `Convert_Outgoing`  
**B** convert only these characters.

**B** *Code conversion with `Convert_Outgoing` (BS2000 systems)*

**B** Code conversion table EBCDIC Code Page EBCDIC.DF.04.1 → 500/1 (EBCDIC)

<b>B</b>	<b>EBCDIC.DF.04.1</b>	<b>EBCDIC</b>
<b>B</b>		<b>Code Page 500/1</b>
<b>B</b>	0x4A	0x79
<b>B</b>	0x4F	0xBB
<b>B</b>	0x5A	0x4F
<b>B</b>	0x5F	0xFF
<b>B</b>	0x6A	0x5F
<b>B</b>	0x79	0xBD
<b>B</b>	0xA1	0xBC
<b>B</b>	0xBB	0x4A
<b>B</b>	0xBC	0xE0
<b>B</b>	0xBD	0x5A
<b>B</b>	0xC0	0xDD
<b>B</b>	0xD0	0x6A
<b>B</b>	0xDD	0xFB
<b>B</b>	0xE0	0xFD
<b>B</b>	0xFB	0xC0
<b>B</b>	0xFD	0xD0
<b>B</b>	0xFF	0xA1

**B** Code conversion with `Convert_Outgoing`

**B** *Code conversion with Convert\_Incoming (BS2000 systems)*

**B** Code conversion table EBCDIC Code Page 500/1 (EBCDIC) → EBCDIC.DF.04.1

<b>B</b>	<b>EBCDIC</b>	<b>EBCDIC.DF.04.1</b>
<b>B</b>	<b>Code Page 500/1</b>	
<b>B</b>	0x4A	0xBB
<b>B</b>	0x4F	0x5A
<b>B</b>	0x5A	0xBD
<b>B</b>	0x5F	0x6A
<b>B</b>	0x6A	0xD0
<b>B</b>	0x79	0x4A
<b>B</b>	0xA1	0xFF
<b>B</b>	0xBB	0x4F
<b>B</b>	0xBC	0xA1
<b>B</b>	0xBD	0x79
<b>B</b>	0xC0	0xFB
<b>B</b>	0xD0	0xFD
<b>B</b>	0xDD	0xC0
<b>B</b>	0xE0	0xBC
<b>B</b>	0xFB	0xDD
<b>B</b>	0xFD	0xE0
<b>B</b>	0xFF	0x5F

**B** Code conversion with Convert\_Incoming

**X/W openUTM conversion tables (Unix, Linux and Windows systems)**

*X/W Code conversion with Convert\_Outgoing (Unix, Linux and Windows systems)*

X/W Code conversion table ISO8859-1 → modified EBCDIC Code Page 500/1  
 X/W (the fields with a gray background indicate modification compared to EBCDIC Code Page  
 X/W 500/1)

X/W		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
X/W	0	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
X/W	1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
X/W	2	40	4F	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
X/W	3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
X/W	4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
X/W	5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	4A	E0	5A	5F	6D
X/W	6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
X/W	7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	6A	D0	A1	07
X/W	8	20	21	22	23	24	25	06	17	28	29	2A	2B	2C	09	0A	1B
X/W	9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
X/W	A	41	AA	B0	B1	9F	B2	BB	B5	BD	B4	9A	8A	BA	CA	AF	BC
X/W	B	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
X/W	C	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
X/W	D	AC	69	ED	EE	EB	EF	EC	BF	80	FD	FE	FB	FC	AD	AE	59
X/W	E	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
X/W	F	8C	49	CD	CE	CB	CF	CC	E1	70	DD	DE	DB	DC	8D	8E	DF

X/W Code conversion with *Convert\_Incoming* (Unix, Linux and Windows systems)

X/W Code conversion table modified EBCDIC Code Page 500/1 (EBCDIC) → ISO 8859-1  
 X/W (the fields with a gray background indicate modification compared to EBCDIC Code Page  
 X/W 500/1)

X/W		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
X/W	0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
X/W	1	10	11	12	13	9D	0A	08	87	18	19	92	8F	1C	1D	1E	1F
X/W	2	80	81	82	83	84	85	17	1B	88	89	8A	8B	8C	05	06	07
X/W	3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
X/W	4	20	A0	E2	E4	E0	E1	E3	E5	E7	F1	5B	2E	3C	28	2B	21
X/W	5	26	E9	EA	EB	E8	ED	EE	EF	EC	DF	5D	24	2A	29	3B	5E
X/W	6	2D	2F	C2	C4	C0	C1	C3	C5	C7	D1	7C	2C	25	5F	3E	3F
X/W	7	F8	C9	CA	CB	C8	CD	CE	CF	CC	60	3A	23	40	27	3D	22
X/W	8	D8	61	62	63	64	65	66	67	68	69	AB	BB	F0	FD	FE	B1
X/W	9	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	E6	B8	C6	A4
X/W	A	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	DD	DE	AE
X/W	B	A2	A3	A5	B7	A9	A7	B6	BC	BD	BE	AC	A6	AF	A8	B4	D7
X/W	C	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
X/W	D	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF
X/W	E	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
X/W	F	30	31	32	33	34	35	36	37	38	39	B3	DB	DC	D9	DA	9F

### 3.2.8 States of a conversation under openUTM

In a conversation, the function calls of both partner programs must be coordinated. A CPI-C program executes a function knowing that the partner program will respond by executing a further function for the same conversation. To aid comprehension of the programming rules on both sides of the conversation, a *state* is introduced for the conversation. The set of follow-up actions within a conversation depends on the respective state of the conversation. Function calls cause the conversation to switch from the current state to another state. This switchover is called a *transition*.

Under openUTM, a conversation can assume the following states:

#### Reset

No conversation is assigned to the conversation\_ID.

#### Initialize

The Initialize\_Conversation call has been terminated successfully and a conversation\_ID was assigned to the conversation.

#### Send

The program is permitted to send data via the conversation (send control).

#### Receive

The program can receive data from the partner via the conversation.

#### Send-Pending

In the last Receive call, the program received data from the partner and simultaneously received send control.

#### Confirm-Send

The local program received a confirmation request and send control from the partner. The local program must respond with a Confirmed call, and then switches to Send state.

With CPI-C programs under openUTM, this state can only occur in conversations with non-UTM applications.

#### Confirm-Deallocate

The local program received a request for receive confirmation from the partner along with the Deallocate notification. The local program must respond with a Confirmed call, and then switches to *Reset* state.

In CPI-C programs under openUTM, this state can only occur in conversations with non-UTM applications.



### Defer-Deallocate

The local program received a request for end of transaction from the partner along with the Deallocate notification. The local program must respond with the corresponding transaction call, and then switches to Reset state.

At the beginning, a conversation is in Reset state. It then enters various follow-up states, depending on the particular function calls and on the information received from the partner. The current state of a conversation can be queried with the `Extract_Conversation_State` call. The state transitions are described in the appendix of the X/Open specification.

If a function is called that is illegal in the currently valid state, the calling program receives the return code `CM_PROGRAM_STATE_CHECK`. The function is not executed.

## 3.3 CPI-C in openUTM

This section contains the following information:

- Under “Supported CPI-C calls” you will find a list of the CPI-C calls available in openUTM.  
Of the calls defined in the X/Open specification on CPI-C, only the most important ones are available in openUTM. Only some of these can be used for communication via the OSI TP protocol.
- The subsection “Restrictions in conversations via the LU6.1 and UPIC protocol” lists the calls that must not be used in conversations with LU6.1 partner applications and openUTM-Clients with the UPIC carrier system.
- “openUTM-specific features of calls” describes the special features that must be noted when calling CPI-C functions under openUTM.
- Under “Behavior when non-supported CPI-C calls are used” you will find a list of the return codes supplied by openUTM if a call that is not available in openUTM is issued in a CPI-C program.
- “Programming rules” describes the rules that must be observed when creating CPI-C programs

### 3.3.1 Supported CPI-C calls

This section provides an overview of the calls available in openUTM at the CPI-C program interface. The overview is in tabular form, whereby the calls are arranged according to task areas. The calls are listed alphabetically in relation to the function name.

The full functionality described here can only be used in conversations to OSI TP partners. In conversations to LU6.1 partners and UTM-Clients with the UPIC carrier system, the calls Confirmed and Send\_Error cannot be used (see [section “Restrictions in conversations via the LU6.1 and UPIC protocol” on page 69](#)).

The CPI-C calls available in openUTM are listed in the following tables.

**Starter set calls**

Function	Call	Description
Accept_Conversation	CMACCP	Accept incoming conversation
Allocate	CMALLC	Set up outgoing conversation
Deallocate	CMDEAL	Terminate conversation (normally)
Initialize_Conversation	CMINIT	Establish outgoing conversation, initialize conversation characteristics
Receive	CMRCV	Receive data
Send_Data	CMSSEND	Send data

**Calls for error and confirmation handling**

Function	Call	Description
Cancel_Conversation	CMCANC	Cancel a conversation
Confirmed	CMCFMD	Send a positive confirmation to the partner
Send_Error	CMSERR	Send error message, negative confirmation

**Calls for data conversion**

Function	Call	Description
Convert_Incoming	CMCNVI	Convert received data from EBCDIC format into the character set used on the local system
Convert_Outgoing	CMCNVO	Convert the data to be sent from the character set used on the local system into EBCDIC format

**Calls for querying information on conversation characteristics**

Function	Call	Description
Extract_Conversation_State	CMECS	Query the current state of the conversation
Extract_Conversation_Type	CMECT	Query the current value of the characteristic conversation_type
Extract_Maximum_Buffer_Size	CMEMBS	Query the maximum length of the data that can be sent or received
Extract_Partner_LU_Name	CMEPLN	Query the name of the current partner (value of the characteristic partner_LU_name)
Extract_Sync_Level	CMESL	Query the current value of the characteristic sync_level
Extract_TP_Name	CMETPN	Query the current value of the characteristic TP_name (name of the current partner program / transaction code)

**Calls for modification of conversation characteristics**

Function	Call	Description
Set_Conversation_Type	CMSCT	Set a value for the characteristic conversation_type
Set_Deallocate_Type	CMSDT	Set a value for the characteristic deallocate_type
Set_Partner_LU_Name	CMSPLN	Set a value for the characteristic partner_LU_name
Set_Receive_Type	CMSRT	Set a value for the characteristic receive_type
Set_Send_Type	CMSST	Set a value for the characteristic send_type
Set_Sync_Level	CMSSL	Set a value for the characteristic sync_level
Set_TP_Name	CMSTPN	Set a value for the characteristic TP_name

### 3.3.2 Restrictions in conversations via the LU6.1 and UPIC protocol

The calls CFMD (Confirmed) and CMSERR (Send\_Error) must not be used in conversations with LU6.1 partners and UTM-Clients with the UPIC carrier system:

#### Confirmed (CMCFMD)

This call is used for sending a receive confirmation. It is only permitted if the conversation is in Confirm-Send or Confirm-Deallocate state, i.e. if a confirmation request has been received from the remote partner. This cannot occur in conversations to LU6.1 partners and UTM client applications with the UPIC carrier system. In this case, the Confirmed call is rejected with the return code CM\_PROGRAM\_STATE\_CHECK.

#### Send\_Error (CMSERR)

This is used to respond negatively to the request for receive confirmation (sending of a negative confirmation).

A Send\_Error call in a conversation to an LU6.1 partner or an UTM-Client with the UPIC carrier system results in a program abort and the loss of all conversations being held by the program at this time. In this case, the effect of the Send\_Error call corresponds to that of a Cancel-Conversation call.

### 3.3.3 openUTM-specific special features of CPI-C calls

This section describes the openUTM-specific special features of the CPI-C calls available in openUTM. The calls themselves are described with the associated error and return codes in X/Open Preliminary Specification: “The CPI-C Specification Version 2 (1994)”. Knowledge of this specification is essential for creating CPI-C programs.

#### **Accept\_Conversation (CMACCP)**

Accepts an incoming conversation, i.e. a conversation initiated by a partner.

CMACCP can only be called once within a program run of the CPI-C program under openUTM. If the program code of the CPI-C program contains more than one CMACCP call, the sequencing logic of the program must ensure that the CMACCP call is executed only once in a program run.

If the CPIC program is started by openUTM with a dialog transaction code, then CMACCP must be called exactly once in this program run.

If the CPIC program is started by openUTM with an asynchronous transaction code, then the CMACCP call can be omitted. In this case, however the conversation cannot be accessed, i.e. no data can be received from the initiator.

If CMACCP is called a second time in a program run, the call is rejected with the return code `CM_PROGRAM_STATE_CHECK`. See also [section “Multiple conversations in one CPI-C program” on page 54](#).

**Allocate (CMALLC)**

Establish an outgoing conversation to a partner. The following must be noted for this call:

- A maximum of six open outgoing conversations is permitted within a program run.
- The maximum number of parallel OSI TP associations/LU6.1 sessions that can exist between the local UTM application and a remote partner application is limited. This is defined in the KDCDEF generation. If all associations or sessions are taken up with other conversations when the Allocate call is issued, the call is rejected with the return code `CM_ALLOCATE_FAILURE_NO_RETRY`.
- The maximum number of associations and sessions that can be maintained simultaneously by the local UTM application depends on the generation.  
If the maximum number of associations and sessions is already reached with other conversations at the time of the Allocate call, the call is rejected with the return code `CM_ALLOCATE_FAILURE_NO_RETRY`

The CMALLC call is mapped to KDCS APRO call. Return codes are converted as follows :

KDCS return code	CPI-C return code
APRO 40Z KD10	<code>CM_ALLOCATE_FAILURE_RETRY</code>
APRO 40Z remaining DC-Codes	<code>CM_ALLOCATE_FAILURE_NO_RETRY</code>
APRO 44Z KD04	<code>CM_PRODUCT_SPECIFIC_ERROR</code>
APRO 44Z remaining DC-Codes	<code>CM_PARAMETER_ERROR</code>
APRO 46Z	<code>CM_PARAMETER_ERROR</code>
APRO remaining CC-Codes	<code>CM_PRODUCT_SPECIFIC_ERROR</code>

**Cancel\_Conversation (CMCANC)**

Abort a conversation. In openUTM, the program run and all conversations that exist at the time are aborted.

The call is mapped to `PEND FR`, i.e. the program is terminated with errors and the local transaction is rolled back. If the CPI-C trace is activated with level `DUMP` or `ALL`, the call is aborted internally with `PEND ER` and a dump is created. In the event of serious errors, it may happen that openUTM initiates a `PEND ER` dump itself even though the job or environment variable `CPICDUMP` is not set. See also [section “Error diagnosis in CPI-C programs” on page 100](#).

Calling `Cancel_Conversation` may result in the loss of messages which have already been sent to the initiator.

### Confirmed (CMCFMD)

Results in the transmission of a receive confirmation (positive confirmation).

This call is not permitted in a conversation with an LU6.1 or UTM client partner with the UPIC carrier system.

It is supported in openUTM in order to respond to the confirmation request of a remote partner on a non-UTM system. CPI-C programs under openUTM cannot request confirmations (the statement `sync_level=CM_Confirm` is not permitted under openUTM). If both partner programs of the conversation are running under UTM, the Confirmed call has no effect.

### Convert\_Incoming (CMCNVI)

Convert the data received with CMRCV (Receive) into the local character set.

CMCNVI assumes that the data transferred in the *buffer* parameter is present in character set EBCDIC Code Page 500/1 format, and converts it into the characters set used by the local system.

**B** On BS2000 systems, CPI-C assumes that the character set used locally is EBCDIC.DF.04.1.

**X/W** On Unix, Linux and Windows systems, CPI-C assumes that the character set used locally is ISO 8859-1, i.e. the 8-bit code Western European (latin 1) which is compatible with ASCII.

With a heterogeneous link, the sender and the receiver should convert the data. With an homogenous link, no conversion is necessary. However, it is essential that both sender and receiver do not convert. See [section “Converting characteristics and user data” on page 57](#).

### Convert\_Outgoing (CMCNVO)

Convert the data into the EBCDIC Code Page 500/1 character set before it is sent with CMSEND (Send\_Data).

CMCNVO assumes that the data transferred in the *buffer* parameter is present in the character set used locally and converts it into character set EBCDIC Code Page 500/1.

**B** On BS2000 systems, CPI-C assumes that the character set used locally is EBCDIC.DF.04.1.

**X/W** On Unix, Linux and Windows systems, CPI-C assumes that the character set used locally is ISO 8859-1, i.e. the 8-bit code Western European (latin 1) which is compatible with ASCII.

With a heterogeneous link, the sender and the receiver should convert the data. With an homogenous link, no conversion is necessary. However, it is essential that both sender and receiver do not convert. See [section “Converting characteristics and user data” on page 57](#).



### **Deallocate (CMDEAL)**

Terminate a conversation and release the conversation ID.

If the conversation characteristic `deallocate_type` has the value `CM_DEALLOCATE_SYNC_LEVEL` or `CM_DEALLOCATE_FLUSH`, the following applies:

- In an asynchronous conversation, only the initiator (client) can terminate the conversation normally and call `CMDEAL`.
- In a dialog conversation, only the acceptor (server) can terminate the conversation normally and call `CMDEAL`. In addition, the acceptor must send at least one message with the `Send_Data` call to the initiator between the transition of the conversation to `Send` state or `Send-Pending` state and the `CMDEAL` call.

See also [section “Programming rules” on page 83](#).

If the conversation characteristic `deallocate_type` has the value `CM_DEALLOCATE_ABEND`, a `Deallocate` call will terminate the calling program with all conversations it is holding at that time. The `Deallocate` call then has the same effect as the `Cancel_Conversation` call.

### **Extract\_Conversation\_State (CMECS)**

Query the current state of the conversation.

There are no restrictions on this call in openUTM.

### **Extract\_Conversation\_Type (CMECT)**

Query the current value of the conversation characteristic `conversation_type`.

With CPI-C under openUTM, the result of this call can only ever be the value `CM_MAPPED_CONVERSATION`.

**Extract\_Maximum\_Buffer\_Size (CMEMBS)**

Query the maximum length of the data that can be sent with Send\_Data call or received with the Receive call.

In openUTM, the maximum buffer size is set to 32767 bytes (as proposed by X/Open). If a value greater than the buffer size received through the call is specified in CPI-C calls for the parameters `buffer_length`, `request_length`, or `send_length`, the corresponding CPI-C call is rejected with `CM_PROGRAM_PARAMETER_CHECK`.

The following must thus apply:

$$0 \leq \dots\_length \leq CM\_MAXIMUM\_BUFFER\_SIZE$$

**Extract\_Partner\_LU\_Name (CMEPLN)**

Query the current value of the conversation characteristic `partner_LU_name`.

With incoming conversations, the call returns the name of the partner application. In openUTM, the name of the LTERM partner is returned for conversations via the UPIC protocol, the name of the LPAP partner is returned for communication via LU6.1, and the name of the OSI-LPAP partner is returned for communication via OSI TP (see [page 47f](#)).

With outgoing conversations, the call only returns the name of the partner application if the conversation characteristic `partner_LU_name` has previously been set with the Set\_Partner\_LU-Name call. Otherwise, blanks are returned.

**Extract\_Sync\_Level (CMESL)**

Query the current value of the conversation characteristic `sync_level`.

There are no restrictions on this call in openUTM.

**Extract\_TP\_Name (CMETPN)**

Query the current value of the conversation characteristic `TP_name`.

With an incoming conversation, the call returns the transaction code with which the local program was started. If you defined several transaction codes for a CPI-C program, you can use CMETPN to determine the transaction code with which the program was started.

With outgoing conversations, the LTAC name of the partner, which was set using the CMSTPN (Set\_TP\_Name) call, is returned. If `TP_Name` was not explicitly called for the conversation, the value transferred with the CMINIT (Initialize\_Conversation) in the `sym_dest_name` parameter is returned.

## Initialize\_Conversation (CMINIT)

Initialize the conversation characteristics of an outgoing conversation.

The symbolic partner name *sym\_dest\_name* must be generated as the LTAC name in the KDCDEF generation of the application (see also [section “Conversation characteristics for addressing” on page 47](#)). Whether *sym\_dest\_name* is a valid LTAC name is not checked until Allocate is called.

A maximum of six open outgoing conversations is permitted at any one time within a program run. If the CPI-C program is holding six open outgoing conversations and if an attempt is made with CMINIT to initialize a seventh conversation, then the call is rejected with the return code CM\_PRODUCT\_SPECIFIC\_ERROR (see also [section “Multiple conversations in one CPI-C program” on page 54](#)).

## Receive (CMRCV)

Receive data from partner

In conversations with *sync\_level* = CM\_SYNC\_POINT or *sync\_level* = CM\_SYNC\_POINT\_NO\_CONFIRM the value CM\_TAKE\_COMMIT, or CM\_TAKE\_COMMIT\_DEALLOCATE or CM\_TAKE\_BACKOUT, or CM\_TAKE\_BACKOUT\_DEALLOCATE is returned for the parameter *status\_received*.

In openUTM, the Receive call is not permitted in Send-Pending state. In Send state, the call is only permitted if CMSEND (Send\_Data) was already called in this state or if Send state has come from Send\_Pending state.

If a CPI-C server program under openUTM transfers send control to the remote partner, i.e. if it calls CMRCV in Send state or following a CMSEND with *send\_type* = CM\_SEND\_AND\_PREP\_TO\_RECEIVE, it may happen that the program does not return from this Receive call and is aborted by openUTM.

This may be caused by the following:

- The logical connection to the partner via which the conversation is established has been lost.
- The partner terminated the conversation abnormally.
- The maximum time for which the local CPI-C program unit can wait for a message from the partner has expired without the partner sending a message. This maximum wait time is defined by the generation parameter MAX PGWTTIME=.

openUTM writes a message in the SYSLOG file with the reason for the termination of the program run.

The Receive call is mapped to the KDCS call MGET. If the return code with MGET is not equal to 000, 01Z, 03Z, 10Z or 12Z, the program is terminated with PEND ER abnormally.

**Send\_Data (CMSEND)**

Send data to partner.

The CMSEND call is mapped to the FPUT KDCS call. If the return code resulting from FPUT is not equal to 000, the CPI-C return code CM\_PRODUCT\_SPECIFIC\_ERROR is generated and the UTM error 40Z is logged with K704 in CPI-C trace.

**Send\_Error (CMSERR)**

Send an error message to the remote partner.

The call may only be used in a dialog conversation.

This call is not permitted in conversations via the LU6.1 or UPIC protocol.

**Set\_Conversation\_Type (CMSCT)**

Set the conversation characteristic conversation\_type.

The call parameter conversation\_type can only have the value CM\_MAPPED\_CONVERSATION. The value CM\_BASIC\_CONVERSATION is not supported by openUTM and is rejected with the return code CM\_PRODUCT\_SPECIFIC\_ERROR.

**Set\_Deallocate\_Type (CMSDT)**

Set the conversation characteristic deallocate\_type.

For the deallocate\_type parameter, only one of the following values can be specified in openUTM:

- CM\_DEALLOCATE\_SYNC\_LEVEL,
- CM\_DEALLOCATE\_FLUSH, or
- CM\_DEALLOCATE\_ABEND

If CM\_DEALLOCATE\_ABEND is specified, the subsequent Deallocate call results in the termination of the program with all conversations. The effect of this Deallocate call then corresponds to that of a Cancel\_Conversation call.

The parameter value CM\_DEALLOCATE\_CONFIRM is rejected by openUTM with the return code CM\_PRODUCT\_SPECIFIC\_ERROR.

**Set\_Partner\_LU\_Name (CMSPLN)**

Set the conversation characteristic `partner_lu_name`.

The value specified in the *partner\_lu\_name* parameter can be up to 8 bytes long. It must be defined as the LPAP name or the OSI-LPAP name in the KDCDEF generation of the application. If a value greater than 8 bytes is specified in the `partner_lu_name_length` parameter, the call is rejected by openUTM with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

**Set\_Receive\_Type (CMSRT)**

Set the conversation characteristic `receive_type`.

If the conversation is in Receive state, the value of `receive_type` has no significance because a CPI-C program is not called by openUTM until all messages have been received.

The setting of `receive_type` is only effective in openUTM if the Receive call is issued in Send state or Send-Pending state. In this case, a call with `receive_type=CM_RECEIVE_IMMEDIATE` is rejected with the return code `CM_PROGRAM_STATE_CHECK`.

**Set\_Send\_Type (CMSST)**

Set the conversation characteristic `send_type`.

Only the following values can be specified for the call parameter *send\_type* in openUTM:

- `CM_BUFFER_DATA`,  
Send data to the partner without additional information.
- `CM_SEND_AND_PREP_TO_RECEIVE`,  
The data transferred is sent to the partner program immediately; send control is transferred to the partner.
- `CM_SEND_AND_DEALLOCATE`,  
Send data and terminate the conversation normally.

The values `CM_SEND_AND_FLUSH` and `CM_SEND_AND_CONFIRM` are not permitted and are rejected with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

**Set\_Sync\_Level (CMSSL)**

Set the conversation characteristic *sync\_level*.

For the call parameter *sync\_level* only the following values are permitted in openUTM:

- CM\_NONE  
no synchronization is requested for the conversation.
- CM\_SYNC\_POINT\_NO\_CONFIRM  
the conversation is included in a global transaction

The value CM\_CONFIRM is rejected with return code CM\_PRODUCT\_SPECIFIC\_ERROR; the value CM\_SYNC\_POINT is rejected with return code CM\_PARM\_VALUE\_NOT\_SUPPORTED.

**Set\_TP\_Name (CMSTPN)**

Set the conversation characteristic *TP\_Name*.

The value specified for the *TP\_Name* call parameter must be defined as the LTAC name in the UTM application and may not be longer than 8 bytes.

If the length specification in *TP\_name\_length* is greater than 8 bytes, the call is rejected by openUTM with the return code CM\_PRODUCT\_SPECIFIC\_ERROR.

### 3.3.4 Interaction with the TX interface

openUTM supports global transactions in conversations via the TX interface. The TX interface is described in chapter 5.

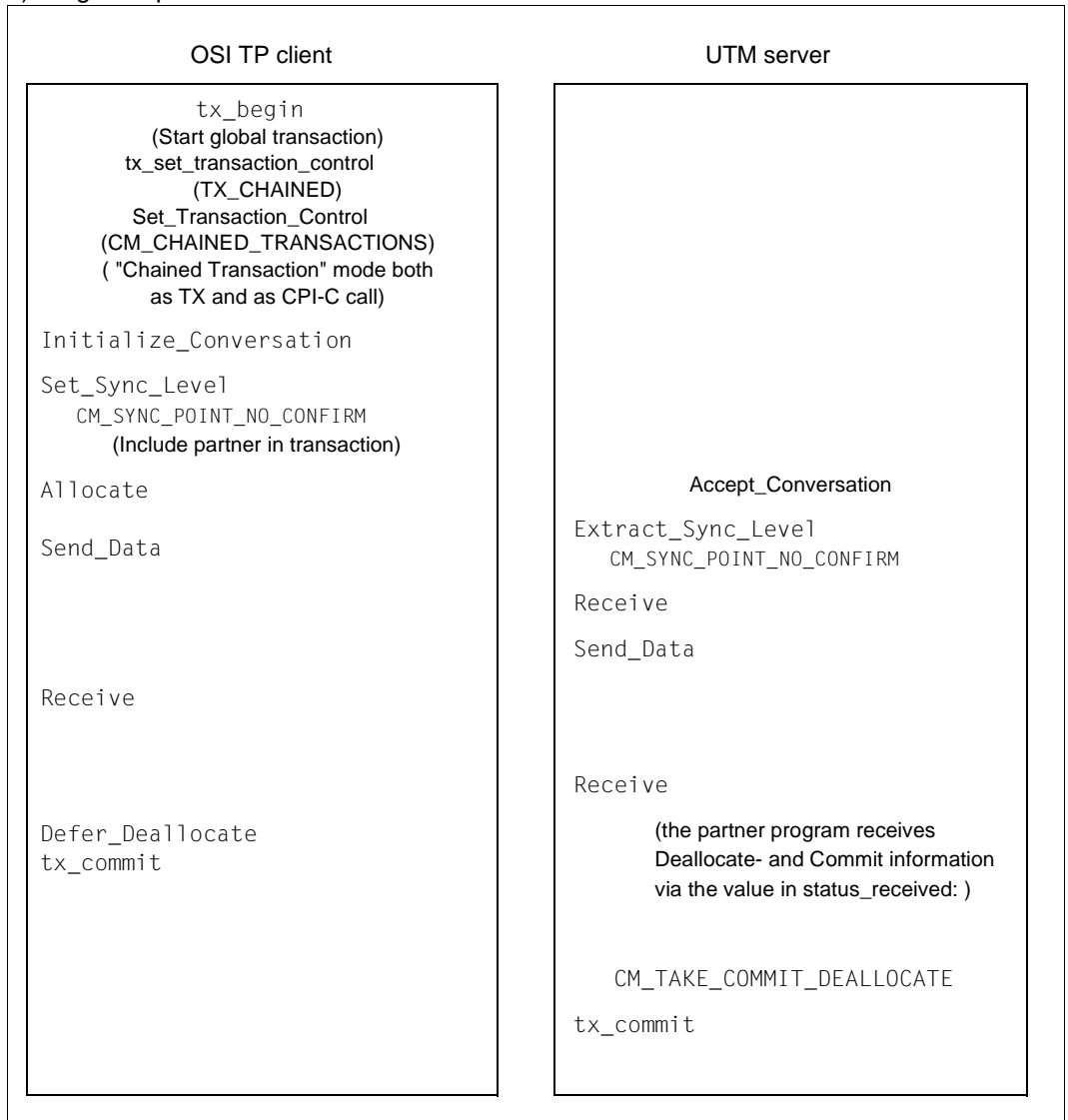


Please note: If you use the TX interface and include UTM server programs in a global transaction, existing UTM server programs behave differently from when they are not included in a global transaction.

In particular, end of transaction may only be requested by the client program!

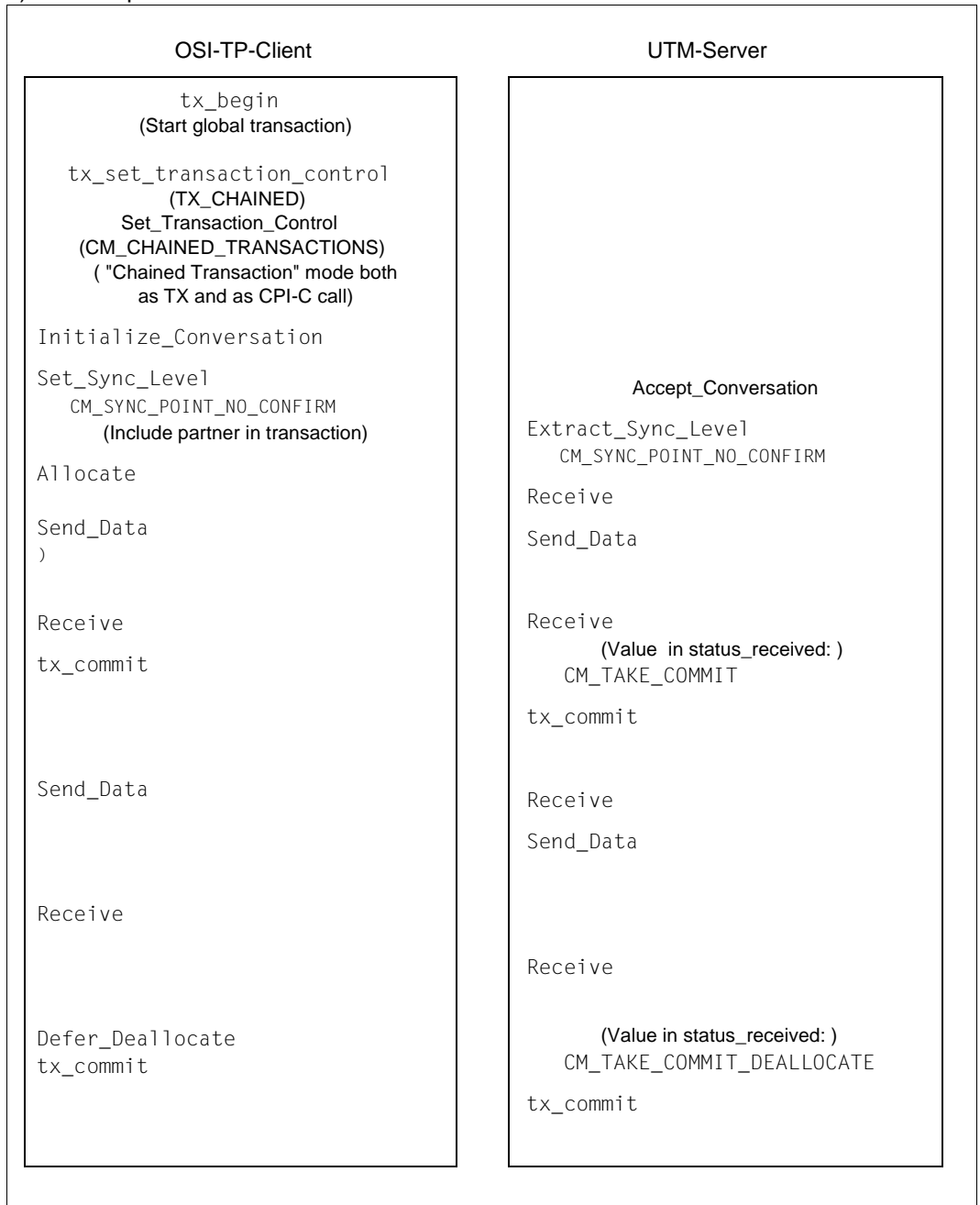
The following diagrams depict the sequence for a single-step and a multiple-step transaction.

## 1) Single-step transaction





## 2) Multi-step transaction



### 3.3.5 Behavior when non-supported CPI-C calls are used

If a CPI-C program under openUTM contains CPI-C calls that are not supported by openUTM, the CPI-C program can be linked with openUTM, but it cannot run under openUTM.

The following calls are rejected with the return code `CM_PRODUCT_SPECIFIC_ERROR`:

Name of function	Call
Confirm	CMCFM
Extract_Mode_Name	CMEMN
Flush	CMFLUS
Prepare_To_Receive	CMPTR
Request_To_Send	CMRTS
Set_Log_Data	CMSLD
Set_Mode_Name	CMSMN
Set_Return_Control	CMSRC
Test_Request_To_Send_Received	CMTRTS

The following call is rejected with the return code `CM_PROGRAM_PARAMETER_CHECK` because it is only permitted with Basic Conversation and openUTM does not support any Basic Conversation.

Name of function	Call
Set_Fill	CMSF

All other optional CPI-C calls not supported by openUTM are rejected with the return code `CM_CALL_NOT_SUPPORTED`.

### 3.3.6 Process or task switching

X/W  
X/W

There is no process switching within a CPI-C program run. A conversation is linked to a process.

B

There is no task switching within a CPI-C program run. A conversation is linked to a task.

### 3.3.7 Programming rules

The rules described below must be observed when creating CPI-C program units under openUTM. The rules refer to conversations with OSI TP partners. Additional restrictions apply in the case of conversations with LU6.1 partners and UTM-Clients with the UPIC carrier system. These restrictions are summarized on [page 69](#).

When creating CPI-C program units, a distinction must be made between *dialog conversations* and *asynchronous conversations*.

With *dialog conversations*, both sides of the conversation can send data, i.e. both the initiator (client) and the acceptor (server). The acceptor of a dialog conversation must be assigned a dialog transaction code in the UTM application. Dialog transaction codes are all generated transaction codes to which the TYPE=D parameter is assigned (default setting).

You must therefore specify the following in the KDCDEF generation:

- TAC tac-name,...[,TYPE=D]  
if the local CPI-C program is the acceptor (server) of the dialog conversation
- LTAC ltac-name,...[,TYPE=D]  
if the partner program is the acceptor (server) of the dialog conversation.

With *asynchronous conversations*, only the initiator (client) of the conversation can send data. The acceptor (server) of an asynchronous conversation must be assigned an asynchronous transaction code in the UTM application. Asynchronous transaction codes are all generated transactions to which the TYPE=A parameter is assigned.

You must therefore specify the following in the KDCDEF generation:

- TAC tac-name,...,TYPE=A  
if the local CPI-C program is the acceptor (server) of the asynchronous conversation
- LTAC ltac-name,...,TYPE=A  
if the partner program is the acceptor (server) of the asynchronous conversation

The structure of a CPI-C program within a conversation depends on whether the conversation is asynchronous or dialog and whether the CPI-C program is the initiator or the acceptor of the conversation.

A distinction must be made between the following cases:

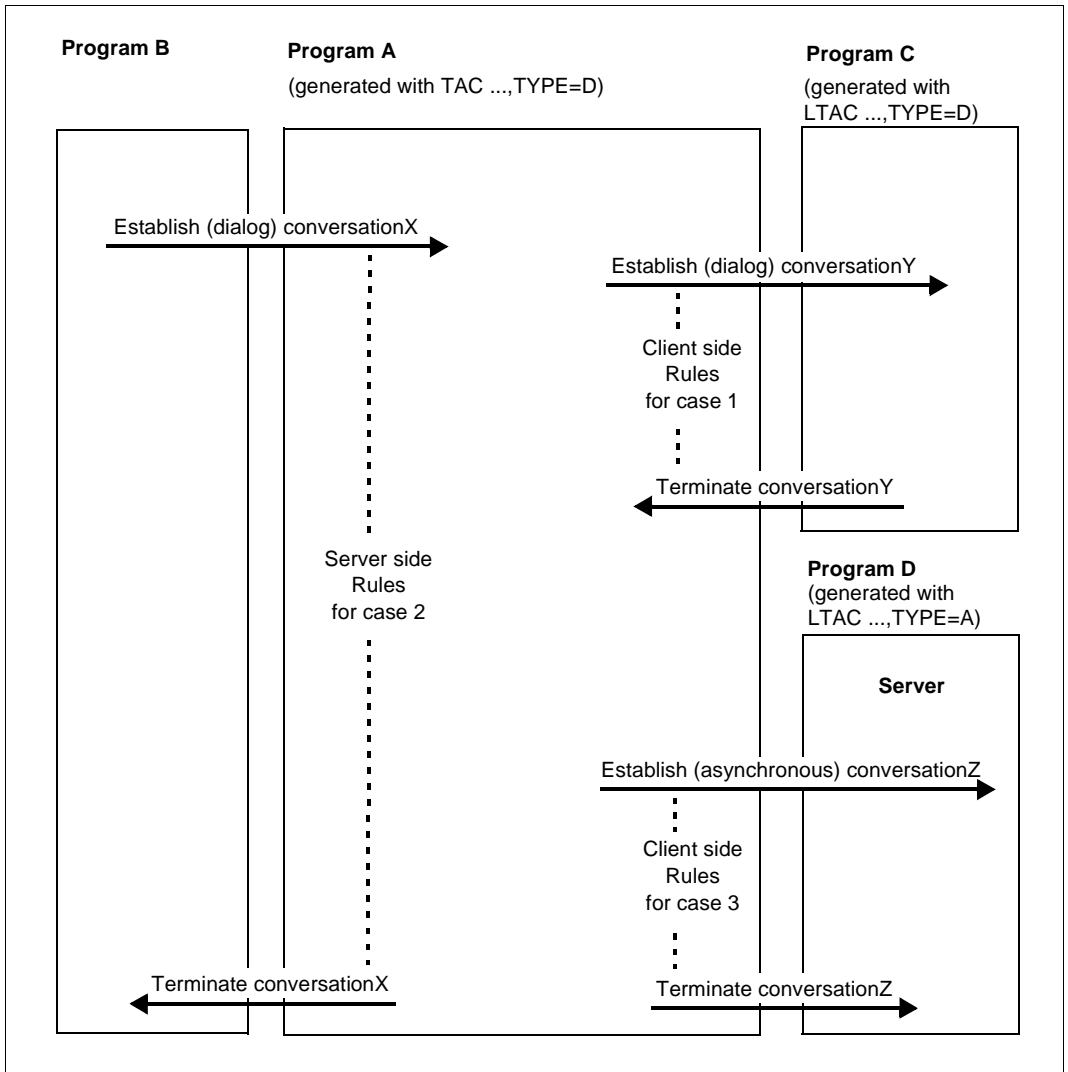
1. The CPI-C program is the initiator of a dialog conversation (outgoing conversation).
2. The CPI-C program is the acceptor of a dialog conversation (incoming conversation).
3. The CPI-C program is the initiator of an asynchronous conversation (outgoing conversation).
4. The CPI-C program is the acceptor of an asynchronous conversation (incoming conversation).

The programming rules for these four cases are listed below. A distinction is made here between the rules for CPI-C programs running under openUTM and CPI-C programs that do not run under openUTM but hold conversations with CPI-C programs under openUTM (such as UTM-Clients that use the CPI-C interface).

The programming rules always apply to one conversation. Each program can hold multiple conversations. A different one of the four cases above may apply to each of these conversations.

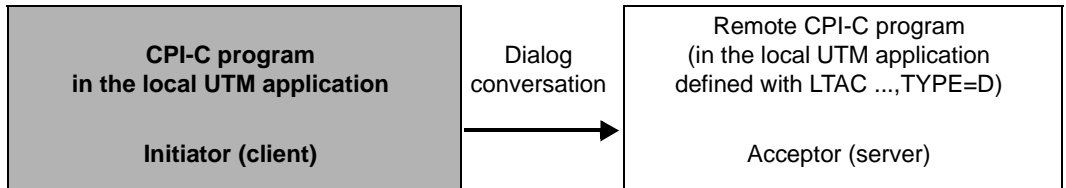
This fact is illustrated in the following diagram by way of an example.

Program A is a CPI-C program under openUTM which is generated using a dialog transaction code. It is started by the request from program B. Program A is thus the acceptor (server) of a dialog conversation. Program A establishes other conversations: a dialog conversation with program C and an asynchronous conversation with program D. In these conversations, program A is the initiator (client).



### Case 1: Initiator of a dialog conversation

The following programming rules apply to the side of the conversation shaded in the diagram.



The following rules apply under openUTM for the initiator of a dialog conversation:

#### Rule 1:

After the conversation is initialized and established, the initiator must send at least one message (possibly of length 0) with the Send\_Data call to the partner program before it is permitted to exit Send state (The initiator switches to Send state after a successful Allocate call).

The following call sequence is thus **not** permitted.

```
Initialize_Conversation
Allocate
Receive
```

#### Rule 2:

In this case, the conversation can only be terminated normally by the acceptor (server), i.e. the initiator can only terminate the conversation abnormally with:

```
Cancel_conversation
```

or

```
Set_Deallocate_Type (deallocate_type=CM_DEALLOCATE_ABEND)
Deallocate
```

However, in this case all conversations held by the program at this time are cancelled under openUTM and the program run is terminated.

The initiator of a dialog conversation must therefore have the following program structure:

```

Initialize_Conversation
.....possibly Set_Partner_LU_Name
.....possibly Set_TP_Name
Allocate
.....possibly Convert_Outgoing
Send_Data
.....possibly further Send_Data
Receive
.....possibly Convert_Incoming
.....possibly further Receive to receive data
.....possibly further sequences of Send_Data and Receive
Receive to wait for return_code CM_DEALLOCATED_NORMAL

```

To query information, the following CPI-C calls can also be issued at any point within a conversation:

- Extract\_Conversation\_State
- Extract\_Maximum\_Buffer\_Size

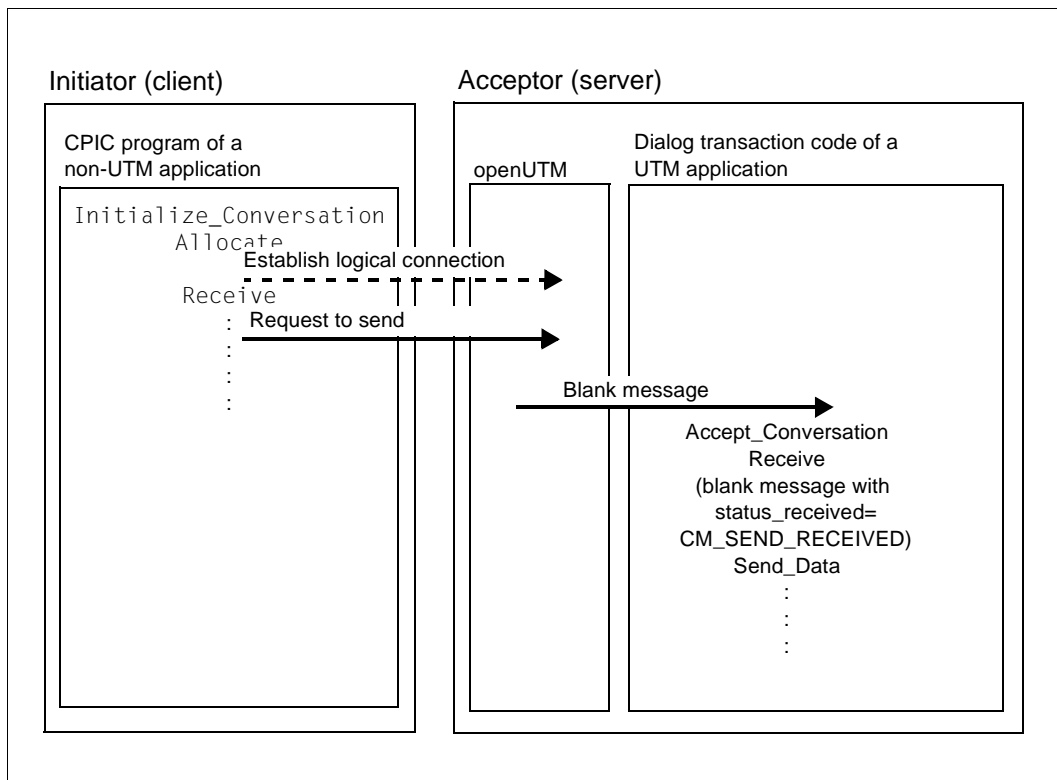
In the event of error, the following CPI-C calls can be used:

- Send\_Error  
(Not permitted with conversations to LU6.1 partners or UTM-Clients with the UPIC carrier system. In this case the call results in a program abort and the loss of all conversations.)
- Cancel\_Conversation
- Set\_Deallocate\_Type with deallocate\_type=CM\_DEALLOCATE\_ABEND followed by Deallocate

If the initiator of the conversation is not running under openUTM (but the acceptor (server) is a CPI-C program under openUTM), rule 1 does not apply.

If the initiator exits the Send state following Allocate but without first sending a message to the acceptor under openUTM with the Send\_Data call, the acceptor receives a message of length zero from openUTM.

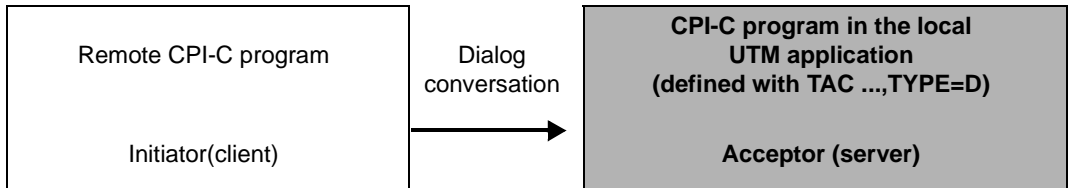
The following program run applies:





## Case 2: Acceptor of a dialog conversation

The following programming rules apply to the side of the conversation shaded in the diagram.



The following rules apply under openUTM for the acceptor of a dialog conversation:

### Rule 1:

`Accept_Conversation (CMA CCP)` must be called as the first function. This call opens the conversation to the initiator.

### Rule 2:

After the `Accept_Conversation` call, the acceptor must continue to read with the `Receive (CMRCV)` call until it receives send control. Thereafter, send control can be switched as often as desired using the `Send_Data (CMSSEND)` and `Receive (CMRCV)` calls.

### Rule 3:

The acceptor must at some time actively close the conversation with the `Deallocate (CMDEAL)` or `Cancel_Conversation (CMCANC)` call, provided the conversation is not terminated abnormally by the initiator or the transport system. An abnormal termination by the initiator or the transport system only occurs if a CPI-C call for this conversation returns with a return code that switches the conversation to Reset state (for information on state transitions, see "X/Open Preliminary Specification CPI-C The Specification Version 2", State Tables in Appendix C).

### Rule 4:

Between entering `Send` state and calling `Deallocate`, the program must send at least one message (possibly with length zero) to the initiator using `Send_Data`, as otherwise the `Deallocate` call will be rejected with `CM_PRODUCT_SPECIFIC_ERROR`.

The following call sequence is therefore **not** permitted:

```
Receive
    CM_SEND_RECEIVED
Deallocate
```

The acceptor of a dialog conversation must therefore have the following program structure:

```
Accept_Conversation
.....possibly Extract_Partner_LU_Name
.....possibly Extract_TP_Name
.....possibly Extract_Sync_Level
Receive
.....possibly Convert_Incoming
.....possibly further Receive to receive data
Receive to wait for status_received=CM_SEND_RECEIVED
.....possibly Convert_Outgoing
Send_Data
.....possibly further sequences of Receive and Send_Data

Deallocate

or

Set_Send_Type (send_type=CM_SEND_AND_DEALLOCATE)
Send_Data
```

You can also invoke the following CPI-C calls at any point within the conversation in order to query information:

- Extract\_Conversation\_State
- Extract\_Maximum\_Buffer\_Size

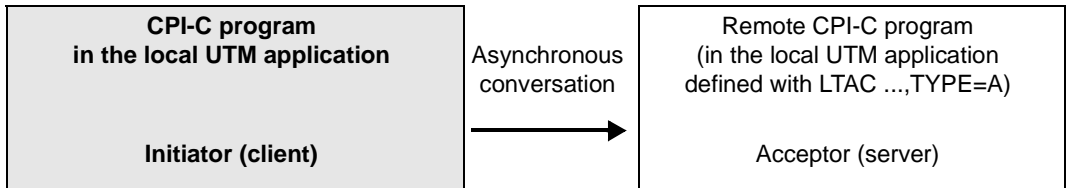
In the event of error, the following CPI-C calls can be used:

- Send\_Error  
(Not permitted in conversations to LU6.1 partners or UTM-Clients with the UPIC carrier system. In this case, it results in a program abort and the loss of all conversations.)
- Cancel\_Conversation
- Set\_Deallocate\_Type with *deallocate\_type*=CM\_DEALLOCATE\_ABEND followed by Deallocate

If the acceptor of the conversation is not running under openUTM, rule 4 does not apply. If the acceptor does not send any message to the initiator running under openUTM between entering *Send* state and issuing the Deallocate call, openUTM sends a message of length zero to the initiator.

### Case 3: Initiator of an asynchronous conversation

The following programming rules apply to the side of the conversation shaded in the diagram.



The following rules apply under openUTM for the initiator of an asynchronous conversation:

#### Rule 1:

The initiator must **not** use the following CPI-C calls:

##### CMCFMD (Confirmed)

Confirmed is only permitted in the states Confirm-Send and Confirm-Deallocate. These states cannot occur at the initiator of an asynchronous conversation because the acceptor cannot send a confirmation request to the initiator.

If the initiator of the conversation is not running under openUTM, this program may still contain the Deallocate call with `deallocate_type=CM_DEALLOCATE_CONFIRM`. However, in this case openUTM does not deliver the confirmation request to the acceptor under its management. Instead, openUTM itself sends a positive confirmation to the initiator as soon as the acceptor (server) of the conversation has accepted all messages.

##### CMRCV (Receive)

The acceptor (server) of an asynchronous conversation does not send any messages to the initiator. A Receive call in the initiator is thus useless and is rejected with `CM_PRODUCT_SPECIFIC_ERROR`.

##### CMSERR (Send\_Error)

The Send\_Error call is rejected with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

##### CMSST (Set\_Send\_Type) with `send_type=CM_SEND_AND_PREP_TO_RECEIVE`

The acceptor of the conversation does not send any messages to the initiator of the conversation. For this reason, any attempt to transfer send control to the acceptor (server) is rejected with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

**Rule 2:**

If both the initiator and the acceptor of the conversation are running under openUTM, the initiator must send at least one message to the acceptor using the `Send_Data` call at least once (possibly with length 0). This means that the acceptor cannot be started using the following simple call sequence:

```
Initialize_Conversation
Allocate
Deallocate
```

If the acceptor is not running under openUTM, this rule does not apply.

The client side of an asynchronous conversation must thus have the following program structure:

```
Initialize_Conversation
.....possibly Set_Partner_LU_Name
.....possibly Set_TP_Name
Allocate
.....possibly Convert_Outgoing
Send_Data
.....possibly further Send_Data

Deallocate
or
Set_Send_Type (send_type=CM_SEND_AND_DEALLOCATE)
Send_Data
```

To query information, the following CPI-C calls can also be issued at any point within the conversation:

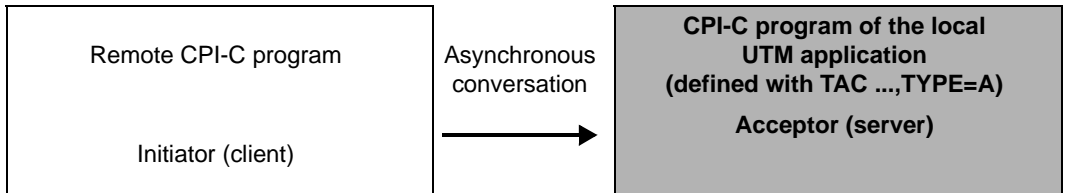
- `Extract_Conversation_State`
- `Extract_Maximum_Buffer_Size`

In the event of error, the following CPI-C calls can be used:

- `Cancel_Conversation`
- `Set_Deallocate_Type` with `deallocate_type=CM_DEALLOCATE_ABEND` followed by `Deallocate`

#### Case 4: Acceptor of an asynchronous conversation

The following programming rules apply to the side of the conversation shaded in the diagram.



The following rules apply under openUTM for the acceptor of an asynchronous conversation:

##### Rule 1:

The acceptor must **not** use the following CPI-C calls:

##### Confirmed (CMCFMD)

Confirmed is only permitted in the states Confirm-Send and Confirm-Deallocate. These states cannot occur at the acceptor of an asynchronous conversation under openUTM for the following reason:

If an initiator not running under openUTM and connected via the OSI TP protocol sends a Deallocate request with a Confirm request to an acceptor of an asynchronous conversation running under openUTM, this is not reported to the CPI-C program under openUTM in the usual way as with a Receive call with `status_received = CM_CONFIRM_DEALLOC_RECEIVED`, rather with `return_code = CM_DEALLOCATED_NORMAL`. openUTM itself sends a positive confirmation to the initiator as soon as the run of the local CPI-C program has ended.

##### Send\_Data (CMSSEND)

The call is rejected with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

##### Send\_Error (CMSERR)

The call is rejected with the return code `CM_PRODUCT_SPECIFIC_ERROR`.

##### Set\_Conversation\_Type (CMSCT)

The call is only permitted in Initialize state. This state cannot occur in this case.

**Rule 2:**

In an asynchronous conversation, only the initiator can terminate the conversation normally. The acceptor under openUTM can only call Deallocate (CMDEAL) if the conversation characteristic `deallocate_type` was previously set to `CM_DEALLOCATE_ABEND` with the `Set_Deallocate_Type` call.

The acceptor (server) of an asynchronous conversation must thus have the following program structure:

```
Accept_Conversation
.....possibly Extract_Partner_LU_Name
.....possibly Extract_TP_Name
.....possibly Extract_Sync_Level
Receive
.....possibly Convert_Incoming
.....possibly further Receive to receive data
Receive to wait for return_code CM_DEALLOCATED_NORMAL
```

To query information, the following CPI-C calls can also be issued at any point within the conversation:

- `Extract_Conversation_State`
- `Extract_Maximum_Buffer_Size`

In the event of error, the following CPI-C calls can be used:

- `Cancel_Conversation`
- `Set_Deallocate_Type` with `deallocate_type=CM_DEALLOCATE_ABEND` followed by `Deallocate`

If the acceptor is not running under openUTM, then it does not finally receive the return code `CM_DEALLOCATED_NORMAL`, it rather receives the value `CM_CONFIRM_DEALLOC_RECEIVED` in the `status_received` parameter. It must then give a positive confirmation with `Confirmed (CMCFMD)` or a negative confirmation with `Send_Error (CMSERR)`. However, a negative confirmation is not delivered to the initiator (i.e. the CPI-C program under openUTM), rather causes the job to be repeated

## 3.4 Creating a CPI-C application

CPI-C program units under openUTM must always be subroutines. openUTM does not transfer any parameters to the CPI-C program unit, i.e. the usual transfer parameters for KDCS program units under openUTM, namely KB, SPAB, and possibly SPAB extensions, must not be used by CPI-C programs.

openUTM then starts a CPI-C program when an incoming conversation for this program is initiated by a partner program.

### 3.4.1 Compiling and linking a CPI-C application on Unix, Linux and Windows systems

X/W CPI-C program units can be created in any programming language.

X/W ● An include file is supplied with openUTM for the creation of CPI-C program units in C:

X – *utmpath/cpic/include/cpic.h* (Unix and Linux systems)

W – *utmpath\cpic\include\cpic.h* (Windows systems)


X/W ● A COPY element is supplied for COBOL

X – *utmpath/cpic/copy-cobo185/CMCOBOL* or *utmpath/cpic/netcobo1/CMCOBOL*  
X (Unix and Linux systems)

W – *utmpath\cpic\copy-cobo185\CMCOBOL* or *utmpath\cpic\netcobo1\CMCOBOL*  
W (Windows systems)

X ● The following XOPEN library is available for linking CPI-C programs on Unix and Linux  
X systems.

X – *utmpath/sys/libxopen*

W  On Windows systems, the X/Open functions are included in the UTM library  
W *libwork.lib*.

X/W ● When you link a CPI-C application, the following libraries must be also be linked:


X/W – the user library with the CPI-C program units and modules

X/W – the UTM library

X – die XOPEN library (Unix and Linux systems)

X/W If you want to create a CPIC trace while the application is running, you can activate the trace  
X/W by means of the start parameters or activate it and deactivate it using the administration  
X/W functions (see [section “Controlling the trace” on page 100](#)).  
X/W

### 3.4.2 Compiling and linking a CPI-C application on BS2000 systems

- B CPI-C program units can be created in any programming language.
- B An include file is supplied with openUTM for the creation of CPI-C program units in C and a COPY element is supplied for COBOL.
- B The library
- B `$userid.SYSLIB..UTM.065.XOPEN` is available for linking CPI-C programs.
- B The include file for C (CPIC.H) and the COPY element for COBOL (CMCOBOL) are also present as type S elements in this library.
- B When linking a CPI-C application, the following libraries must be also be linked:
  - B – the user library with the CPI-C program units and modules
  - B – the UTM library
  - B – the XOPEN library `$userid.SYSLIB.UTM.065.XOPEN`
  - B – the libraries `SYSLNK.CRTE` or `SYSLNK.CRTE.PARTIAL-BIND` for language environment and runtime system.
- B If a CPIC trace is to be created while the application is running, you can activate the trace by means of the start parameters or activate it and deactivate it using the administration functions (see [section “Controlling the trace” on page 100](#)).
- B  Note when changing from openUTM V3.4 to V4.0:  
Use the new library which is used for all XOPEN interfaces when linking!



### 3.4.3 Generating a CPI-C application

The following special features must be noted when generating an UTM application with CPI-C program units:

*Generate local CPI-C program units and associated transaction codes*

Each program unit of an UTM application must be defined with a PROGRAM statement.

**B** On BS2000- systems, you have to specify ILCS as compiler:

**B** PROGRAM programname,COMP=ILCS....

Transaction codes implemented in the CPI-C program units must be generated with

TAC tacname, PROGRAM=programname, API=(XOPEN,CPIC), ...

(where programname=name of the CPI-C program from the PROGRAM statement).

You can assign several transaction codes to a CPI-C program unit. The program run can then be controlled using the transaction code with which the program was started. The transaction code with which the CPI-C program was started is contained in the conversation characteristic TP\_Name.

*Generate TACs for which PGWT is permitted*

If a CPI-C program unit is to hold dialog conversations in which the send control is transferred to the conversation partner by calling Set\_Send\_Type with send\_type=CM\_SEND\_AND\_PREP\_TO\_RECEIVE or by calling Receive in Send state, then, in applications that work with process limit-based job control, the transaction code of this CPI-C program unit must be assigned to a TAC class which is generated with PGWT=YES. For example:

```
MAX TASKS=2
MAX TASKS-IN-PGWT=1
TACCLASS 1,TASKS=1,PGWT=YES
TAC CPIC1,PROGRAM=xyz,API=(XOPEN,CPIC),TACCLASS=1
```

In applications that use priority-based job control, you must generate the transaction code of this CPI-C program unit using PGWT =YES:

```
MAX TASKS=2
MAX TASKS-IN-PGWT=1
TAC CPIC1,PROGRAM=xyz,API=(XOPEN,CPIC), PGWT=YES
```

*Specifications for associations with OSI TP partners and UTM client partners with the OpenCPIC carrier system*

The information required for setting up OSI TP associations between the local application and the partner application must be specified in ACCESS-POINT, OSI-LPAP, and OSI-CON statements as follows, for example:

```
* name of the local access point
ACCESS-POINT  cpicap,\
               P-SEL = NONE, S-SEL = NONE, T-SEL = C'tse',

* local name of remote partner
OSI-LPAP      cpiclpap,\
               ASSOCIATION-NAMES = assoname,\
               ASSOCIATIONS = 2, CONTWIN = 1, CONNECT = 1,\
               APPLICATION-CONTEXT = UDTAC or UDTDISAC or UDTCCR
```



UDTCCR is necessary for transaction-oriented communication (sync-level CM\_SYNC\_POINT\_NO\_CONFIRM).

```
* local name of connection
OSI-CON       cpiccon,\
               P-SEL=NONE, S-SEL=NONE,\
               T-SEL=C'tsel', N-SEL=C'nsel',\
               LOCAL-ACCESS-POINT = cpicap,\
               OSI-LPAP = cpiclpap
```

When communicating with partners on non-SNI systems, it may happen that the Application Entity Title must also be defined for the local and remote application.

*Specifications for sessions with LU6.1 partners*

The information required for setting up LU6.1 sessions between the local application (BCAMAPPL) and the partner application must be specified in BCAMAPPL, LPAP, CON, SESCHA, and LSES statements.

**X/W** *Notes on data conversion*

**X/W** If the calls Convert\_Incoming (CMCNVI) and Convert\_Outgoing (CMCNVO) are used for data conversion, MAP=USER **must** be specified in the OSI-CON statement (for OSI-TP partners) and in the SESCHA statement (for LU6.1 partners) when generating the connections to the partner application.

*Specifications for connections to UTM client partners with the UPIC carrier system*

You must issue a PTERM and an LTERM statement for each UTM client partner with the UPIC carrier system, for example:

```
PTERM UPICTTY, PTYPE=UPIC-R, LTERM=UPICLT, BCAMAPPL=CPICBA, PRONAM=PGTR0175
LTERM UPICLT
```

Alternatively you can also connect via TPOOL.

If a UPIC partner is implementing the user concept of openUTM and therefore transfers security data (user ID, password), USER statements must also be issued for each user ID used.

*Definition of TP names of remote CPI-C programs*

You must issue an LTAC statement for each CPI-C program of a remote application to which a local CPI-C program wants to set up an outgoing conversation.

The LPAP= operand and possibly other operands such as RTAC= or TYPE= must be specified in the LTAC statement. The ltac name of the LTAC statement must be specified in the Initialize\_Conversation call in the sym\_dest\_name parameter.

The above-mentioned KDCDEF statements are described in the openUTM manual "Generating Applications".

## 3.5 Error diagnosis in CPI-C programs

CPI-C calls are mapped to KDCS calls. The UTM dump therefore contains only the KDCS calls, even with CPI-C program units. However, to diagnose CPI-C programs you can create a trace of CPI-C calls in addition to the UTM dump.

### 3.5.1 Controlling the trace

The CPI-C trace can be controlled as follows:

- The UTM start parameter CPIC-TRACE can be used to enable the trace when the application is started, see the relevant openUTM manual “Using UTM Applications”.
- The trace can be enabled or disabled during operation using WinAdmin or WebAdmin. To do this go to the dialog *Properties of UTM Application*, tab *Diagnosis and Account*, field *CPIC Trace*.
- The trace can be enabled or disabled during operation via the KDCADMI administration program interface. This is done using the field *cpic\_trace* in the data structure *kc\_diag\_and\_account\_par\_str*, see openUTM manual “Administering Applications”.

You can set the following trace levels:

Level	Meaning
TRACE	The content of the input and output parameters is output for each CPI-C function call. Only the first 16 bytes are output from the data buffers. The return codes of the KDCS calls to which the CPI-C calls are mapped are output.
BUFFER	Includes the TRACE level. However, the data buffers are logged in their full length.
DUMP	Includes the TRACE level and also writes diagnostic information to the trace file.
ALL	Includes the BUFFER, DUMP and TRACE levels.
OFF	Trace is disabled.

### 3.5.2 Name of the trace file

#### X/W Trace file on Unix, Linux and Windows systems

X/W The trace records are written to a file in the application directory (*filebase*).

X/W – For stand-alone applications, this file has the name `KDC.TRC.CPIC.appliname.pid`.

X/W – For UTM cluster applications, this file has the name  
X/W `KDC.TRC.CPIC.appliname.nodename.pid`.

X/W Where:

X/W appliname

X/W Name of the application.

X/W nodename

X/W Name of the cluster node on which the node application is running.

X/W pid PID of the process.

#### B Trace file on BS2000 systems

B By default, the trace records are written to a file.

B – For stand-alone applications, this file has the name `KDC.TRC.CPIC.appliname.tsn`.

B – For UTM cluster applications, this file has the name  
B `KDC.TRC.CPIC.appliname.nodename.tsn`.

B Where:

B appliname

B Name of the application.

B nodename

B Name of the cluster node on which the node application is running.

B tsn TSN of the UTM task

B In the UTM start procedure, you can also set up a different trace file for each task and use  
B the SET-FILE-LINK command to assign it the link name `KDCCPIC`.

### 3.5.3 Contents of the trace file

The CPI-C calls are logged as follows:

--->IN functionname: input parameters

The responses to the function calls are logged as follows:

<---OUT functionname: return\_code = returncode other output parameters

returncode is the return code of the call. The other output parameters are only output if return\_code = CM\_OK.

In addition, the responses to the KDCS calls are written to the trace file.

---

## 4 X/Open interface XATMI

XATMI has been standardized by X/Open and is a program interface for a communication resource manager which enables transaction-logged client/server communication.

The XATMI program interface is based on the X/Open CAE Specification "Distributed Transaction Processing: The XATMI Specification" of November 1995. Knowledge of this specification is essential for understanding this chapter.

This chapter describes the XATMI interface for programs under openUTM. XATMI programs under openUTM are always server programs.

### Supported protocols

Under openUTM, XATMI applications can communicate via OSI TP or via the LU6.1 protocol.

### Terms

The following terms are used in this description:

Service	<p>A service function that is programmed in C or COBOL in accordance with the XATMI specification.</p> <p>XATMI distinguishes between two different types of services: end Services and intermediate Services.</p> <p>An "end" service is linked only to its client and does not call any other services.</p> <p>An "intermediate" service calls one or more other services.</p>
Client	<p>An application that calls service functions.</p>
Server	<p>An UTM application containing the service functions in C and/or COBOL. The service functions can comprise a number of program units.</p>
Request	<p>A request is a service call. This call can be initiated by a client or by an intermediate service.</p>
Requester	<p>The XATMI specification uses the term "requester" to refer to the application that calls a service. A requester can be either a client or a server.</p>

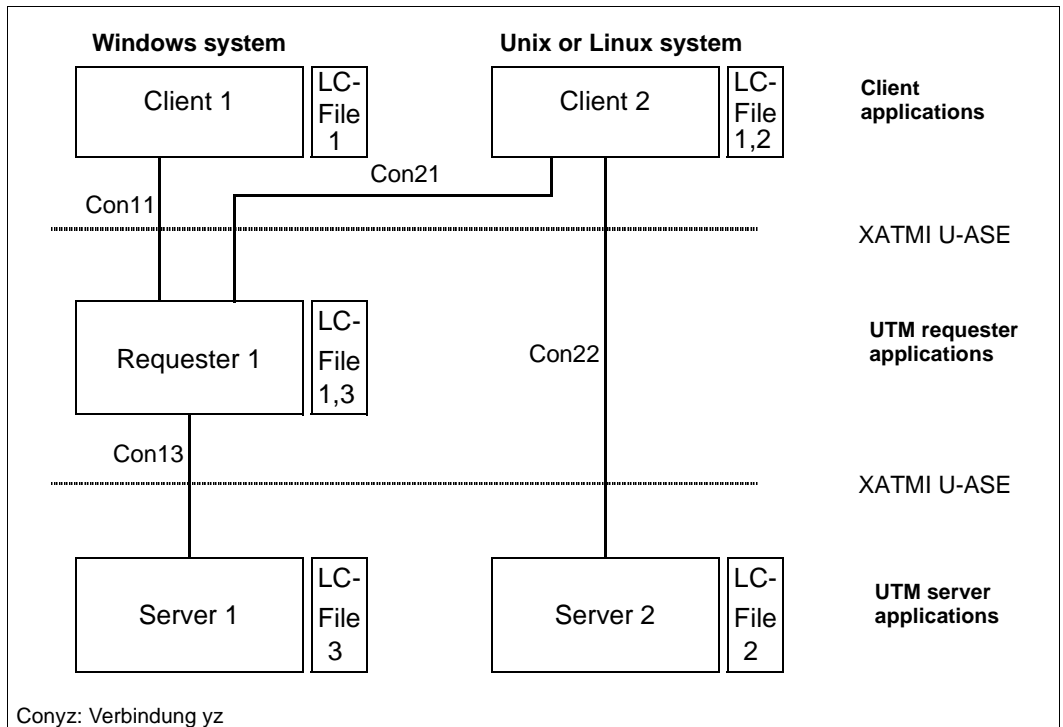
Typed buffers

Buffers for exchanging type-encoded and structured data between communication partners. With these typed buffers, the structure of the exchanged data is implicitly known to the carrier system and the application, and is also adapted automatically (encoded, decoded) in heterogeneous connections.

### 4.1 Linking client/server

The diagram below shows the connection of client/server applications, linking the client, server, and requester. Clients can also be located on BS2000 systems.

The individual applications contain XATMI calls and exchange their type-encoded data structures (**typed buffers**) in accordance with the protocol of the "XATMI U-ASE Definition".





With any heterogeneous application link, a local configuration must be provided both for the servers and the clients. This configuration is defined in the local configuration file (LCF). The local configuration describes the respective services and their associated data structures, namely:

- in the case of a server, all available services
- in the case of a client, the services of all servers to which the client is connected
- in the case of a requester (intermediate service), all services available as well as all services used

The local configurations of all applications involved must be coordinated with each other.

A number of communication paradigms are available for client-server communication (see [section “Communication paradigms” on page 106](#)).

### 4.1.1 Default server

To simplify the client/server configuration openUTM allows you to define a default server using the statement `DEST=.DEFAULT` in the `SVCU` statement of the local configuration file (see [page 122](#)).

You define the default server in the `LTAC` statement of the `KDDCDEF` generation by providing the `LPAP` name for the `SVCU` (here *svcname2*) in the `LPAP` operand, e.g.:

```
LTAC svcname2, ... , LPAP=BRANCH9
```

The partner (in this case *BRANCH9*) must be generated in the `KDCDEF` generation as a partner name in a `OSI-LPAP`, `LPAP`, `MASTER-OSI-LPAP` or `MASTER-LU61-LPAP` statement.

If the calls `tpcall`, `tpacall` or `tpconnect` use a service *svcname2* to which there is no `SVCU` entry in the local configuration file, the following entry is used automatically:

```
SVCU svcname2, RSN=svcname2, TAC=svcname2, DEST=.DEFAULT, MODE=RR
```

## 4.2 Communication paradigms

The programmer can choose from three communication paradigms for client/server communication:

- synchronous request response paradigm: single-step dialog.  
The client is blocked after sending the service request until it receives a response.
- asynchronous request response paradigm: single-step dialog.  
The client is not blocked after sending the service request.

Special case:

Single request paradigm: communication in one direction only.  
The client requests a service, but does not expect a response.

- conversational paradigm: multi-step dialog.  
Client and server can exchange data in any way required.

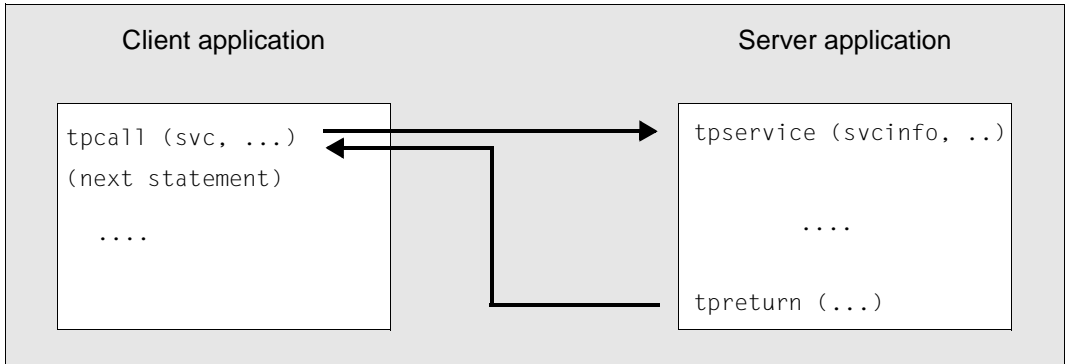
The XATMI functions required for these communication paradigms are described only briefly below; C notation is used here. An exact description of the XATMI functions can be found in the X/Open Specification "Distributed Transaction Processing: The XATMI Specification".

Depending on the paradigm, the communication is started by a *tpcall()*, *tpacall()* or a *tpconnect()* XATMI call.

On each XATMI *tpcall()*, *tpacall()* and *tpconnect()* call in the UTM application, UTM addresses a service (KDCS call APRO) and occupies a session or association for the duration of the communication. It is necessary to generate as many sessions or associations as the maximum number of services that are addressed simultaneously.

### Synchronous request response paradigm

Only a single *tpcall()* call is required for the communication with this type of service. The *tpcall()* call addresses the services, sends precisely one message to this service, and waits until it receives a response, i.e. *tpcall()* has a blocking effect.



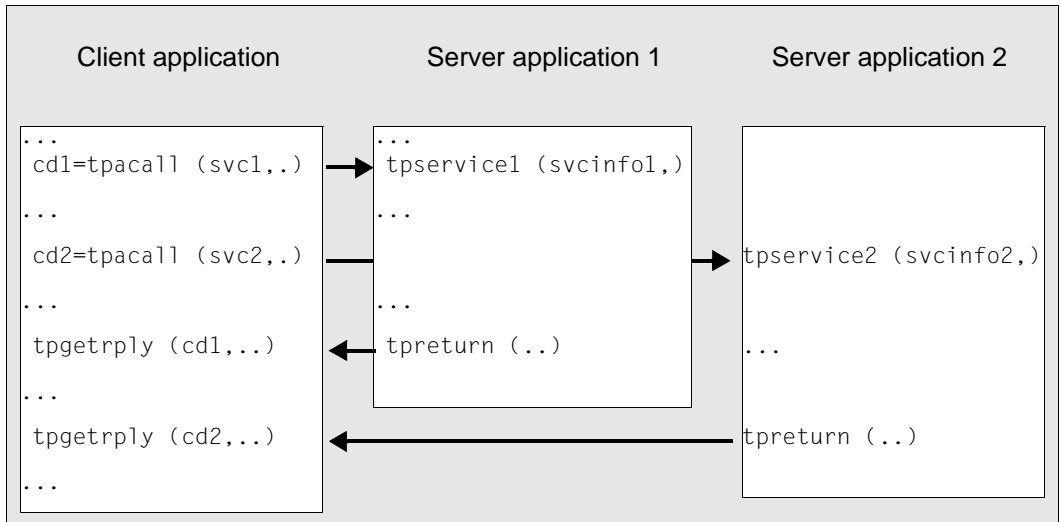
In this diagram, *svc* is the internally used service name, *svcinfo* is the service info structure with the service name, and *tpservice* is the program name of the service routine. The *tpservice()* template is available for this purpose for the programming language C. The XATMI call `TPSVCSTART` must be used for COBOL, see [page 117](#). The service info structure is defined in the XATMI interface and is supplied internally by XATMI.

With this paradigm, a dialog TAC for the requested service has to be generated in the UTM server application.

### Asynchronous request response paradigm

With this paradigm, communication is handled in two steps. In the first step, a *tpacall()* call is used to address the service and send the message. In the second the response is fetched with *tpgetreply()* at a later stage. The *tpacall()* is non-blocking, i.e. the client can continue to perform local processing tasks after the call. Further (up to 64) connections are also possible.

In this paradigm, multiple services can be requested in parallel, see the figure below.



In this diagram, *svc1*, *svc2* refer to the internally used service names, *cd1*, *cd2* are the communication descriptors in the specific process, *svcinfo1*, *svcinfo2* are the service info structures with the service names, and *tpservice1*, *tpservice2* are the program names of the service routines.

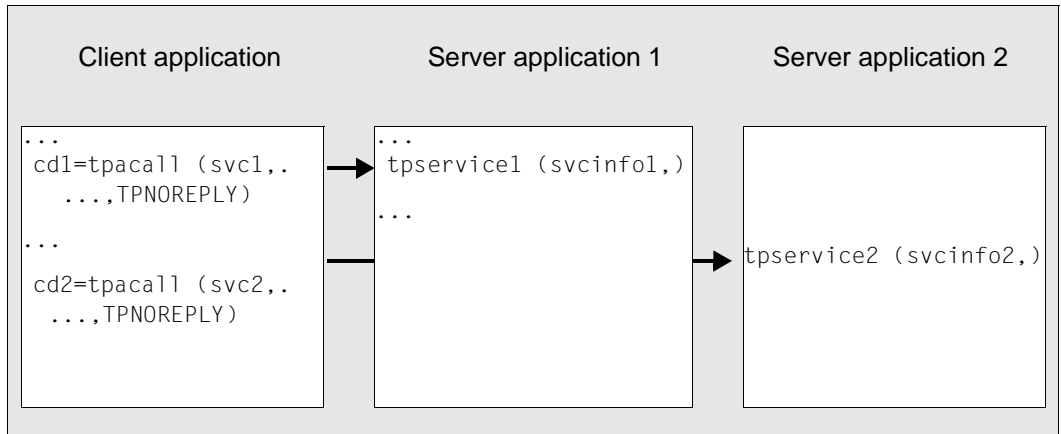
Unlike *tpacall*, *tpgetrply()* is blocking, which means that the client waits until the response is received.

With this paradigm, a dialog TAC must be generated for the requested service in the UTM server application (as with synchronous request response)

Special cases of this paradigm are **single request** and **request with no response**.

### Single request paradigm

With this paradigm, a service is requested with *tpacall()*. However, no response is expected (indicated by the *TPNOREPLY* flag).



In this diagram, *svc1*, *svc2* refer to the internally used service names, *cd1*, *cd2* are the communication descriptors in the specific process, *svcinfo1*, *svcinfo2* are the service info structures with the service names, and *tpservice1*, *tpservice2* are the program names of the service routines.

With this paradigm, an asynchronous TAC must always be generated in the addressed UTM server application.

This paradigm can only be used if the client is a UTM requester application.

### Conversational paradigm

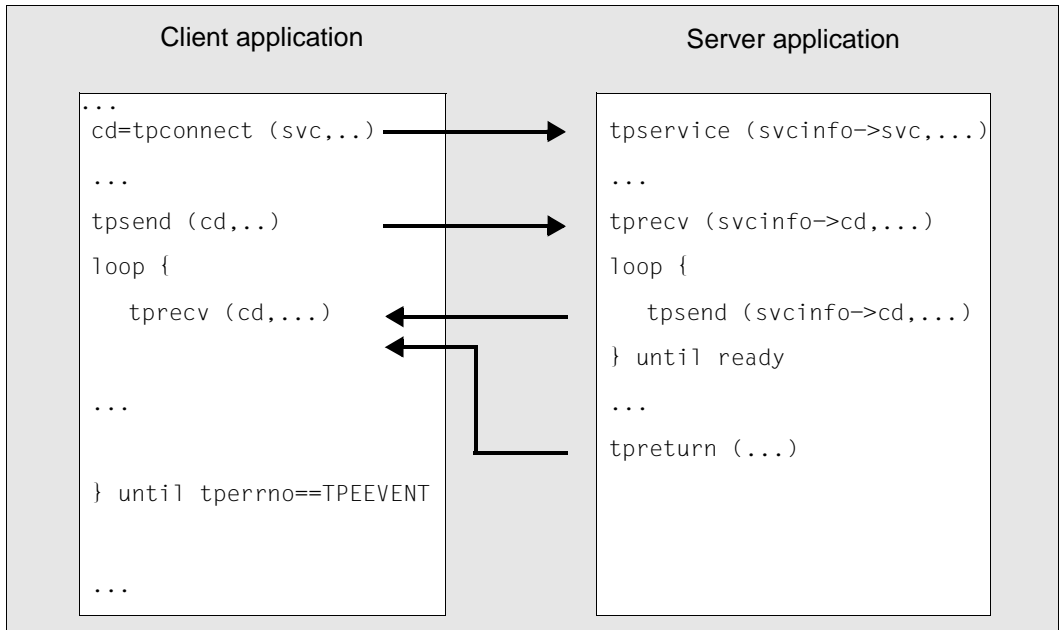
XATMI offers the conversational paradigm for connection-oriented tasks ("conversation").

This paradigm can be used, for example, to transfer large volumes of data in several substeps. This avoids problems which can occur in the synchronous request response paradigm (function *tpcall()*) due to the limited size of the data buffers.

In the conversational paradigm, the conversation is explicitly established to a service with the *tpconnect()* call. As long as the conversation exists, the client and server can exchange data with *tpsend()* and *tprecv()*. Only one transaction can be handled within a dialog.

The conversation is terminated when the server signals the end with *tpreturn()*; the client then receives a corresponding code with *tprecv()* in the *tperrno* variable.

The client program must therefore contain at least one *tprecv()* call.



In this diagram, *svc* refers to the local name of the service, *cd* is the communication descriptor in the specific process, *tpservice* is the program name of the service routine, and *svcinfo* is the service info structure with the service name and the communication descriptor.

With this paradigm, a dialog TAC must be generated for the requested service in the UTM server application.

In the event of errors, the client can force a conversation abort with the *tpdiscon()* call.

### 4.3 Typed buffers

XATMI applications exchange messages using “typed data buffers”. This ensures that the data sent over the network is transferred correctly between applications, i.e. in accordance with the data structure - and associated data types - which is identified by the buffer name.

The advantage of this is that the applications need not take account of any machine dependencies, such as Big Endian/Little Endian representations, ASCII/EBCDIC conversions, or alignments with word limits. This means that data types such as int, long, float, etc. can be

transferred as such. There is no need for any encoding/decoding by the applications because this is carried out by XATMI (in accordance with the rules of the XATMI U-ASE definition).

A data buffer object comprises four components:

- type: defines the type of buffer; there are three types (see below)
- subtype: defines the object of the class, i.e. the actual data structure
- length specification
- data contents

This type of data buffer is created at runtime and can then be addressed by its variable name (= subtype name). In C programs, these buffers are created dynamically with *tpalloc()* and are then called “typed buffers”; in COBOL programs, these buffers are defined statically and are then called “typed records”.

## Types

The data buffer type defines which elementary data types of the employed programming language are permitted. This enables a shared data understanding in a heterogeneous client/server network.

Three types are defined in XATMI:

X_OCTET	Non-typed data stream of bytes (“user buffer”). This type has no subtypes. No conversion takes place.
X_COMMON	All data types that can be used in common by C and COBOL. Conversion is carried out by XATMI.
X_C_TYPE	All elementary C data types, with the exception of pointers. Conversion is carried out by XATMI.

## Subtypes

A subtype has a name of up to 16 characters, with which it is addressed in the application program. Each subtype is assigned a data structure (C structure or COBOL record) which determines the syntax of the subtype, see [page 119](#).

The data types must not be nested.

The structure of a subtype is represented by a syntax string in the local configuration. In this string each elementary data type (basic type) is identified by a code which, if necessary, may also contain the field length specification (<m> and <n>).

The table below provides an overview of the elementary data types (basic types), their codes, and the character set of the string types::

Code <sup>1</sup>	Meaning	ASN.1 type	X_C_TY P E	X_COM MON
s	short integer	INTEGER	short	S9(4) COMP-5
S<n>	short integer array	SEQUENCE OF INTEGER	short[n]	S9(4) COMP-5 ...
i	integer	INTEGER	integer	-- <sup>2</sup>
I<n>	integer array	SEQUENCE OF INTEGER	integer[n]	--
l	long integer	INTEGER	long	S9(9) COMP-5
L<n>	long integer array	SEQUENCE OF INTEGER	long[n]	S9(9) COMP-5 ...
f	float	REAL	float	--
F<n>	float array	SEQUENCE OF REAL	float[n]	--
d	double	REAL	double	--
D<n>	double array	SEQUENCE OF REAL	double[n]	--
c	character	OCTET STRING	char	PIC X
t	character	T.61-String	char	PIC X
C<n>	character array: All values from 0 thru 255 (decimal)	OCTET STRING	char[n]	PIC X(n)
C!<n>	character array, termi- nated by null ('\0')	OCTET STRING	char[n]	--
C<m>:<n>	character matrix <sup>3</sup>	SEQUENCE OF OCTET STRING	char [m][n]	--
C!<m>:<n>	character matrix, termi- nated by null ('\0')	SEQUENCE OF OCTET STRING	char [m][n]	--
T<n>	The printable characters A-Z, a-z, and 0-9 plus <sup>4</sup> a range of special characters and control characters, see <a href="#">page 148</a> .	T.61 string	char[n]	PIC X(n)
T!<n>	character array, termi- nated by null ('\0')	T.61-String	t61str[n]	--
T<m>:<n>	character matrix	SEQUENCE OF T.61-String	t61str[m][n]	--



Code <sup>1</sup>	Meaning	ASN.1 type	X_C_TYP E	X_COMMON
T!<m>:<n>	character matrix, terminated by null ('\0')	SEQUENCE OF T.61-String	t61str[m][n]	--

<sup>1</sup> used in the local configuration to describe the data structures

<sup>2</sup> -- : not available in X\_COMMON

<sup>3</sup> character matrix: twodimensional character array

<sup>4</sup> in accordance with CCITT Recommendation T.61 or ISO 6937

The assignment between data structures, subtypes, and desired services is defined in the local configuration, see [page 122](#).

### Character set conversion with X\_C\_TYPE and X\_COMMON

The data buffers are transmitted over the network encoded in the ASCII character set.

However, a partner can use a character set encoding other than ASCII. For example a BS2000 application may use EBCDIC. In this case, the BS2000 XATMI library converts the ASN.1 type *T.61 string* from ASCII to EBCDIC and vice versa for all incoming and outgoing data. This means that no automatic conversion may be generated for the associated carrier system (UTM/UPIC/OpenCPIC).

X/W  
X/W  
X/W

In particular, MAP = USER must be specified in the OSI-CON statement (for OSI TP partners) or in the SESCHA statement (for LU6.1 partners) when generating connections to the partner application.

The OCTET STRING data type is not converted.

## 4.4 Program interface

The following sections provide an overview of the XATMI program interface for the server and requester. A detailed description of the program interface as well as the error and return codes can be found in the X/Open specification "Distributed Transaction Processing: The XATMI Specification". Knowledge of this specification is essential for creating XATMI programs.

The program interface is available in both C and COBOL.

### 4.4.1 XATMI functions

The following tables list all XATMI calls permitted under openUTM and indicate the role in which they can be called (C = Client or S = Server) and the communication paradigm with which they are permitted

**Calls of the request/response paradigm**

<b>C call</b>	<b>COBOL call</b>	<b>Call in Client/ Server</b>	<b>Description</b>
tpcall	TPCALL	C	Service request in synchronous request/response paradigm
tpacall	TPACALL	C	Service request in asynchronous request/response paradigm or single request paradigm (flag TPNOREPLY set)
tpgetreply	TPGETRPLY	C	Response request in synchronous request/response paradigm
tpcancel	TPCANCEL	C	deletes an asynchronous service request before the requested response is received

**Calls for the conversational paradigm**

<b>C call</b>	<b>COBOL call</b>	<b>Call in Client/ Server</b>	<b>Description</b>
tpconnect	TPCONNECT	C	establishes a connection for message exchange
tpsend	TPSEND	C, S	sends a message
tprecv	TPRECV	C, S	receives a message
tpdiscon	TPDISCON	C	closes down a connection for message exchange

**Calls for typed buffers**

<b>C call</b>	<b>COBOL call</b>	<b>Call in Client/ Server</b>	<b>Description</b>
tpalloc	--	C, S	reserves memory area for a typed buffer
tprealloc	--	C, S	modifies the size of a typed buffer
tpfree	--	C, S	releases a typed buffer
tptypes	--	C, S	ascertains the type of a typed buffer

**General calls for service routines**

<b>C call</b>	<b>COBOL call</b>	<b>Call in Client/Server</b>	<b>Description</b>
(tpservice)	TPSVCSTART	S	makes available a template for service routines
tpreturn	TPRETURN	S	ends a service routine
tpadvertise tpunadvertise	TPADVERTISE TPUNADVERTISE	S	only supported syntactically in openUTM (ascertains the name of a service routine)

## 4.4.2 Particularities of XATMI calls

This section describes the special features of XATMI calls under openUTM.

### **tpcancel**

When communication is performed using transport protocol LU6.1, the *tpcancel()* call to an existing `call descriptor` does not, unlike to X/Open CAE Specification for XATMI, delete this descriptor. Instead, it ends the conversation, i.e. all connections are reset when *tpreturn()* is called. The XATMI calls are mapped to the generation-dependent RSET and PEND FR under openUTM.

When communication is performed using transport protocol OSI TP, *tpcancel()* deletes the descriptor. The XATMI calls are mapped to CTRL AB under openUTM.

### **tpdiscon**

When communication is performed using transport protocol LU6.1 (with or without Commit), the *tpdiscon()* call to an existing `call descriptor` does not, unlike to X/Open CAE Specification for XATMI, delete this descriptor. Instead, it ends the conversation, i.e. all connections are reset when *tpreturn()* is called. The XATMI calls are mapped to the generation-dependent RSET and PEND FR under openUTM

When communication is performed using transport protocol OSI TP without Commit *tpdiscon()* has the same effect as with communication via LU6.1.

When communication is performed using transport protocol OSI TP with Commit *tpreturn()* calls the KDCTXCOM or KDCTXRLB dialog TACs with PEND KP in the UTM server application. In the case of a single request service (*tpacall* TPNOREPLY) PEND FR or PEND FI is called.

## 4.4.3 Data transfer to the service function

A service function is a program unit that is started by a transaction code, as is usual in openUTM. This transaction code must be generated with TAC... API=(XOPEN,XATMI) and can only be called by an XATMI client or requester, not from a terminal.

The client/requester information is transferred to the service by means of the "service info structure" TPSVCINFO. This structure is part of the XATMI interface. Transfer to the service is different in C and COBOL:

- In C, the only parameter which the service function contains is a pointer to the service info structure. This structure is supplied immediately after the service function is started. The *tpservice()* template for the service function is described in the X/OPEN XATMI specification.

TPSVCINFO has the following structure:

```
typedef struct {
    char name[XATMI_SERVICE_NAME_LENGTH]; /* service name */
    char *data; /* pointer to data area */
    long len; /* length of data buffer transferred */
    long flags; /* flags for call control*/
    int cd; /* conversation descriptor in
            conversational mode, otherwise
            undefined */
} TPSVCINFO
```

In COBOL, the structure is supplied by the XATMI call TPSVCSTART. This call must be the first XATMI call of a service routine. Further details can be found in the X/Open specification “Distributed Transaction Processing: The XATMI Specification” in the description of TPSVCSTART.

#### 4.4.4 Events and error handling

When an event or an error occurs, XATMI functions return the return code -1. The program must evaluate the *tperrno* variable to determine the event or error more precisely.

With the conversational function *tprecv*, *tperrno=TPEEVENT* indicates that an event has occurred. This event can be determined by evaluating the *tprecv* parameter *revent*. For example, the successful termination of a conversational service is indicated as follows:

```
Return code of tprecv == -1
tperrno=TPEEVENT
revent=TPEV_SVCSUCC
```

The *revent* parameter is of no significance with the *tpsend* function.

Furthermore, at the end of the service function the service program can return a freely defined error code with *tpreturn* in the *rcode* parameter; this error code can be evaluated on the client side using the external variable *tpurcode*, see “Distributed Transaction Processing: The XATMI Specification”.

### 4.4.5 Creating typed buffers

Typed buffers are defined by data structures in include files (in C) or COPY elements (in COBOL), which must be used in the participating programs.

Data is exchanged between the programs on the basis of these data structures, which must therefore be known to both the client and the server. All data types described in the table on [page 112](#) or in the X/Open specification “Distributed Transaction Processing: The XATMI Specification” may be used.

The header files or COBOL COPY files in which the typed buffers are described serve as input for the generation program `xatmigen`, see [page 128](#) for further information.

The following rules apply to these files:

- C and COBOL data structures must be contained in separate files. A file that contains both C includes and COBOL COPY elements is not permitted as input.
- The files can only contain definitions of data structures, blank lines, and comment statements. Include files (in C) may also contain Macro statements, i.e. statements beginning with ‘#’.
- The data structure definitions must be specified in full. In particular, COBOL data records must begin with the level number “01”.
- The data structures must not be nested.
- Only absolute values are permitted as field lengths, macro constants are not accepted.
- Only the data types listed in the table on [page 112](#) are permitted. In particular, no pointer types are permitted in C.

The user may have to use the generation tool `xatmigen` to map the character arrays to ASN.1 string types because neither C nor COBOL recognizes these data types; see [section “The xatmigen utility” on page 128](#).

XATMI calls for memory allocation are available for C (*tpalloc ...*).

Two simple examples are provided below for C and COBOL respectively.

## Example

### 1. C include for typed buffer

```
typedef struct {
    char   name[20];    /* person's name */
    int    age;         /* age */
    char   sex;
    long   shoesize;
} t_person;

struct t_city {
    char   name[32];    /* name of city */
    char   country;
    long   inhabitants;
    short  churches[20];
    long   founded;
}
```

### 2. COBOL COPY for typed record

```
***** Personal record
01 PERSON-REC.
   05 NAME      PICTURE      X(20).
   05 AGE       PIC S9(9) COMP-5.
   05 SEX       PIC X.
   05 SHOESIZE PIC S9(9) COMP-5.

***** City record
01 CITY-REC.
   05 NAME      PIC X(32).
   05 COUNTRY   PIC X.
   05 INHABITANTS PIC S9(9) COMP-5.
   05 CHURCHES  PIC S9(4) COMP-5 OCCURS 20 TIMES.
   05 FOUNDED   PIC S9(9) COMP-5.
```

Further examples can be found in the X/Open XATMI Specification.



#### 4.4.6 Characteristics of XATMI in openUTM

This section describes the distinctive features that arise when implementing the XATMI interface in openUTM.

- All XATMI calls are supported. However, the two calls *tpadvertise()* and *tpunadvertise()* are only supported syntactically because they are process- or task related and these calls are ineffective in the BS2000 multitask environment or the openUTM work process environment on Unix, Linux and Windows systems.
- An intermediate service can call up to 64 services in parallel via the Asynchronous Request-Response Model or the Conversational Model.
- A maximum of 100 buffer entities can be used simultaneously per service. For example, with a service function in C this is a maximum of 100 *tpalloc()* calls without *tpfree()* call.
- The maximum message length is 32000 bytes. However, this value is also limited by the following parameters in UTM generation:

MAX NB: maximum length of logical messages

The maximum size of a typed buffer is always less than the maximum possible message length because the messages contain an “overhead” in addition to the net data. The more complex the buffer, the bigger the overhead.

The following applies as a rule of thumb: max. buffer size = 2/3 of max. message length

With larger data volumes, the conversational paradigm (*tpsend/tprecv*) should thus always be used.

- The following limits apply to name lengths:

service name: 16 bytes

buffer name: 16 bytes

In accordance with the standard, service names can be 32 bytes long (`XATMI_SERVICE_NAME_LENGTH` constant) where only the first 16 bytes are relevant. It is therefore advisable to use no more than 16 bytes for service names.

- Process switching (Unix, Linux and Windows systems) or task switching (BS2000 systems): A service or request is linked to a process or task, i.e. there is no process or task switching within a service or request. Requests and conversational services must therefore run within one PGWT-TAC class (see [section “Generating TACs for which PGWT ist permitted” on page 131](#)).

## 4.5 Configuring

The user must create a local configuration for each XATMI application. This describes the services provided and used, together with their target addresses, and also describes the typed buffers used with their syntax. The information is stored in a file, known as the local configuration file (LCF), which is read once by the application at startup. An LCF is required both for the client and the service side.

### 4.5.1 Creating the local configuration file

As users, you must create an input file known as the local configuration definition file. This input file must be made up of individual lines that comply with the following syntax:

- A line begins with an SVCU, SVCP, REQP, or BUFFER statement. The following are specified with these statements:
 

SVCU	service used
SVCP	service offered
REQP	request offered
BUFFER	subtype (=typed buffer)
- Two operands are separated by a comma.
- A statement is concluded by a semicolon (;).
- If the operands occupy more than one line, the continuation character '\ ' (backslash) must appear at the end of each line.
- A comment line begins with the '#' character in column 1.
- Blank lines can be inserted, e.g. to improve legibility.

Using the `xatmigen` tool, you create the actual local configuration file ([page 128](#)) from the file which contains the local configuration definition.

The four statements are described below.

#### **SVCU statement: Define available service**

In an SVCU statement, the characteristics required to call a service in the partner application are described for the client/requester. You must specify an SVCU statement for each service used if no default server is used.

You do not have to specify the SVCU statement if XATMI works with the UPIC carrier system and a default server is specified in the UPIC Sideinformation file for which *transaction-code = remote-service-name = internal-service-name* and if the default settings are sufficient for the remaining parameter values.

*Default-Server:*

To simplify the client server configuration openUTM allows to define a default server in the local configuration file by entering `DEST=.DEFAULT` in the SVCU statement. You find more details and an example in [section "Default server" on page 105](#).

Operator	Operand	Explanation
SVCU	<code>internal-service-name</code> <code>[,RSN=remote-service-name]</code> <code>[,TAC=transaction-code]</code> <code>,DEST=destination-name</code>  <code>[,MODE=<u>RR</u> / RN / CV]</code>  <code>[,BUFFERS=(subtype-1,...,subtype-n)]</code>	maximum 16 bytes default: internal-service-name default: internal-service-name partner application RR=request/response, default RN=request no response CV=conversation default: no subtype

**internal-service-name**

A name of up to 16 bytes under which a (remote) service can be addressed in the *local* application. This name must be unique within the application.

Multiple definitions are not verified. The first *internal-service-name* is valid, any others of the same name are ignored.

Mandatory operand!

**RSN=remote-service-name**

A name of up to 16 bytes of a service in the *remote* application. This name is transferred to the remote application (TPSVCINFO structure); it can appear repeatedly in the LCF.

If this operand is omitted, the tool `xatmiggen` sets the value *internal-service-name* for RSN.

**TAC=transaction-code**

A transaction code of up to 8 bytes with which the service is addressed in the local application. You can use the utility program `xatmiggen` to convert this transaction code into an LTAC statement for KDCDEF generation. The corresponding naming conventions in openUTM apply.

If this operand is omitted, the tool `xatmiggen` sets the value *internal-service-name* for TAC and, if necessary, truncates this to the first 8 bytes.

**DEST= destination-name**

A partner application identification of up to 8 bytes.

You must generate this name as *lpapname* in an OSI-LPAP or LPAP statement or in a MASTER-OSI-LPAP or MASTER-LU61-LPAP statement.

.DEFAULT      A default server is used.

mandatory operand!

MODE=RR / RN / CV

Determines which communication paradigm is used for the service:

RR	request response paradigm (default value)
RN	request with no response paradigm (single request), initiates the start of an asynchronous TAC on the openUTM side
CV	conversational paradigm

BUFFERS=(subtype-1,...,subtype-n

List of subtype names that can be sent to the service (the type X\_OCTET is always permitted). Each name can be up to 16 bytes long.

A separate BUFFER statement, which defines the characteristics of the particular subtype, must be specified for each of the subtypes listed here (see BUFFER statement [page 126](#)).

The BUFFERS= operand is sensitive to position and must (if specified) be the *last* operand of the statement.

If BUFFERS= is omitted, only a buffer of type X\_OCTET should be sent to the service (no type verification is performed).

### SVCP and REQ statement: Define service/request offered

- With an SVCP statement, the characteristics of an available end service are described for the server. This service does not call any other services. PGWT calls must be permitted for an end service with the conversational paradigm (see [page 131](#)).
- With an REQ statement, the characteristics of an available intermediate service are described for the server. This service in turn calls another service. For such services PGWT calls must be permitted (see [page 131](#)).

SVCP and REQP statements are optional serve generation to simplify UTM generation for the user. If they are specified, you can use the tool `xatmigen` to create the suitable KDCDEF statements for the services, see [page 128](#) and [page 131](#). Otherwise, users must do this themselves.

Operator	Operand	Explanation
SVCP REQP	<code>internal-service-name</code> <code>[,TAC=transaction-code]</code> <code>[,PROG=program-name]</code> <code>[,COMP=compiler-language]</code>  <code>[,MODE=<u>RR</u> / RN / CV]</code>	maximum 16 bytes default: <code>internal-service-name</code> default: <code>internal-service-name</code> default: C (on Unix, Linux and Windows systems) ILCS (unter BS2000 systems) RR=request/response, default RN=request no response CV=conversation

**SVCP** The statement 'Service Provided' defines end services.

**REQP** The statement 'Request Provided' defines intermediate services.

**internal-service-name**

A name of up to 16 bytes under which a (remote) service is addressed in the *local* application. This name must be unique within the application.

Multiple definitions are not verified. The first *internal-service-name* is valid, others of the same name are ignored.

Mandatory operand!

**TAC=transaction-code**

A transaction code of up to 8 bytes with which the service program unit is started. The naming conventions of openUTM apply.

You can use the utility program `xatmigen` to generate a communication model-dependent TAC statement for KDCDEF generation, possibly with an associated TACCLASS statement.

If this operand is omitted, `xatmigen` sets `TAC=internal-service-name` and, if necessary, truncates this to the first 8 bytes.

**PROG=program-name**

A name of up to 8 bytes for the program unit that provides the service function. This name is converted into a PROGRAM statement. You can use the utility program `xatmigen` to convert this name into a PROGRAM statement for KDCDEF generation.

The naming conventions of openUTM apply.

**COMP=compiler-language**

Describes the programming language of the service program unit and is converted into the COMP= operand in the PROGRAM statement:

X/W C C program, default value on Unix, Linux and Windows systems

X/W COB2/MFCOBOL/NETCOBOL  
X/W COBOL program

X/W CPP C++ program

B ILCS Linkage via ILCS, default value on BS2000 systems

B Only COMP=ILCS may be used on BS2000 systems.

**MODE=RR / RN / CV**

Determines which communication paradigm is used for the service:

RR request response paradigm, default value.

RN request with no response paradigm (single request). An asynchronous TAC is generated in this case

CV conversational paradigm

**BUFFER statement**

A BUFFER statement defines a typed buffer. Buffers of the same name must be defined in the same way on both the client side and the server side.

Multiple definitions are not verified. The first buffer entry is valid, all others are ignored.

Buffers of type "X\_OCTET" have no special features and therefore do not require any definition. Typed buffers are defined using the following parameters:

Operator	Operand	Explanation
BUFFER	subtype-name [, REC=referenced-record-name] [, TYPE=X_COMMON / X_C_TYPE]	maximum 16 bytes default: subtype-name default: xatmigen sets TYPE automatically

**subtype-name**

A buffer name of up to 16 bytes which must also be specified in the BUFFERS= operand in the SVCU statement. The name must be unique in the application.

**REC=referenced-record-name**

Name of the data structure for the buffer, e.g. with C structures this is the name of "typedef" or the "struct name".

If the operand is omitted, xatmigen sets REC=subtype-name.

TYPE=

Type of buffer; for further details on types see [page 111](#).

If the operand is omitted, `xatmi gen` sets the type to `X_C_TYPE` or `X_COMMON`, depending on which elementary data types were used.

TYPE is ignored if neither `X_COMMON` or `X_C_TYPE` is set for TYPE or if the data structure is not of the specified buffer type.

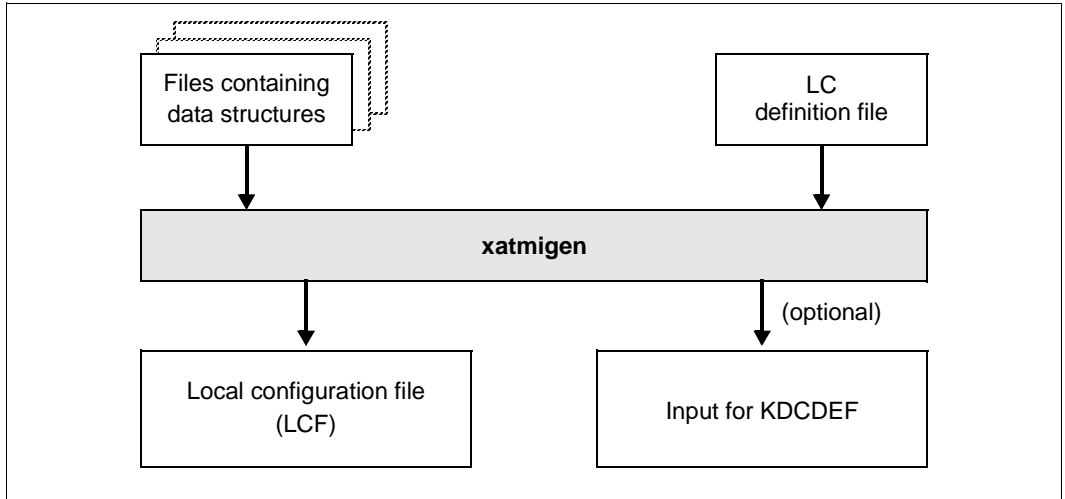
In the generation run, the `xatmi gen` tool also creates two additional operands with the following meaning:

LEN=length                      length of the data buffer

SYNTAX=code                    syntax description of the data structure in code representation, as specified in the table on [page 112](#).

## 4.5.2 The xatmigen utility

The `xatmigen` utility creates a local configuration file (LCF) from a file containing the local configuration definition (LC definition file) and one or more files containing C or COBOL data structures (LC description files), see diagram below:



The local configuration file is structured in the same way as the LC definition file, and differs from this only in the additional descriptions of the buffer type, buffer length, and buffer syntax string. In other words, the operands `LEN=`, `SYNTAX=`, and possibly `TYPE=` are added to the `BUFFER` statements compared to the definition file.

If the buffer type is not specified in the LC definition file, `xatmigen` generates the “smallest” value range for the buffer type, i.e. first the type `X_COMMON`.

All file names must be specified explicitly. If desired, a file can be created which contains the generation statements for `KDCDEF`.

**X/W** Success and error messages are written to `stdout` and `stderr` on Unix, Linux and Windows systems.

**B** Success and error messages are written to `SYSOUT` and `SYSLST` on BS2000 systems.

Although in principle it is possible to edit the LCF, you are strongly advised not to do this.



## Calling xatmigen

- X/W** On Unix, Linux and Windows systems, `xatmigen` is called with
- X/W** `xatmigen parameter`
- X/W** `xatmigen` can be found in the following directory:
- X** `utmpath/xatmi/ex` (Unix and Linux systems) or
- W** `utmpath\xatmi\ex` (Windows systems)
- B** On BS2000 systems, you start `xatmigen` with the SDF command `START-XATMIGEN`.
- B** Alternatively, you can also start `xatmigen` using the following command:
- B** `/START-EXECUTABLE-PROGRAM FROM-FILE=*LIB`
- B** `ELEM(LIBRARY=$userid.SYSLNK.UTM.065.UTIL, ELEMENT-OR-SYMBOL=XATMIGEN)`

You can specify the following parameters; the switches (`-d`, `-l`, `-i`, `-c`) must be written in lowercase:

```
[utm]
  -d _lcdf-name
  -l _lcf-name]
  -i]
  -c _stringcode]
  [_descript-file-1] . . . [_descript-file-n]
```

**utm** If “utm” is specified, `xatmigen` creates a file with generation statements for KDCDEF from the SVCU, SVCP, and REQ statements. `xatmigen` only generates TAC and TACCLASS statements for applications with process limitation-based job control. The generation statements created by `xatmigen` must be supplemented before the UTM generation, see [page 131](#).

If specified, `utm` must always be the first `xatmigen` parameter. If it is not specified, no generation statements are created.

the generated file has the name `xtutm.def`. The file is written into the current directory (Unix and Linux systems) or into the current user ID (BS2000 systems).

**-d \_lcdf-name**  
Name of the LC definition file; mandatory specification.

**-l \_lcf-name**  
Name of the local configuration file to be created. The name must comply with the conventions of the operating system in question.

If the LCF is used simultaneously for a Unix, Linux and Windows system oder for a BS2000 and Windows system, it is advisable to choose a name with a maximum of 8 characters and add the extension “.lcf”.

Any existing LCF of the same name is automatically overwritten.

If the switch is omitted, `xatmigen` creates the `xatmilcf` file in the current directory (Unix-/Windows system) or in the current user ID (BS2000 system).

- i Interactive mode: the string code is queried for each typed buffer containing a character array. The possible specifications for the string code are described under the “-c” switch.

The -i switch takes priority over a -c switch, if this is specified.

If `xatmigen` is running in the background or in batch mode, the -i switch must not be specified.

#### -c\_stringcode

the specified string type applies for the entire `xatmigen` run, i.e. for all character arrays. In interactive mode (“-i”), the “-c” switch is ignored.

The following can be specified for *stringcode* (see table on [page 112](#)):

C octet string  
 C! octet string, terminated by '\0'  
 T T.61 string  
 T! T.61 string, terminated by '\0'

If no value is specified, T! is used.

Single characters are also interpreted as T.61 string (*stringcode*= t).

#### descript-file-1 . . . \_descript-file-n

List of files containing the include files or COPY elements with the data structures of the typed buffers.

If this statement is not present, only buffer type X\_OCTET is permitted.

#### Note

The -d switch and, if specified, the -l and -c switches must each be followed by the associated parameter. Specification of the switch without this parameter is not permitted.

### 4.5.3 KDCDEF generation

For an XATMI application to be functional, you must carry out the following steps:

- define the program units of the services offered locally (TAC and PROGRAM statements), where the following applies:
  - in applications with priority-based control, PGWT calls must be permitted for these TACs
  - in applications with process limitation, these TACs must be assigned to TAC classes for which PGWT is permitted
- define the services used on the remote system (LTAC statements)
- generate the OSI TP or LU6.1 connections

#### 4.5.3.1 Generating TACs for which PGWT ist permitted

A service is always linked to a work process (Unix, Linux and Windows systems) or task (BS2000 systems). As soon as both requests and conversational services are contained in a server application, at least two work processes or tasks must be started and a TAC class for which PGWT calls must be permitted for the TAC or the TAC must be assigned to a TAC class for which PGWT is permitted.

This is not necessary for an application that contains only request/response services.

#### 4.5.3.2 Number of sessions (LU6.1) or associations (OSI TP)

With each XATMI call *tpcall()*, *tpacall()* and *tpconnect()* a service is addressed in openUTM (KDCS APRO call). This means that a separate session or association must be generated for each of these XATMI calls.

#### 4.5.3.3 Defining the services offered

The following KDCDEF statements must be specified for the services offered:

- A PROGRAM and TAC statement for each program unit that provides a service function. The TAC statement must contain the operand `API=(XOPEN,XATMI)`.

If "utm" is specified as the *carrier system* parameter in the `xatmi gen` utility, a PROGRAM statement and a TAC statement are created for each SVCP or REQ statement. These statements can be used in unchanged form as KDCDEF input.

**Example:**

```

PROGRAM svcproc1, COMP = ILCS
TAC      service1, PROGRAM = svcproc1, API = (XOPEN,XATMI)
PROGRAM svcproc2, COMP = ILCS
TAC      service2, PROGRAM = svcproc2, API = (XOPEN,XATMI)

```

- The corresponding statements must be specified for each connection to a remote partner:
  - The statements ACCESS-POINT, OSI-LPAP, and OSI-CON for OSI TP connections to an openUTM or OpenCPIC partner, e.g. as follow:

```

* name of local access point
ACCESS-POINT  server,
              P-SEL = *NONE, S-SEL = *NONE, T-SEL = C'tsel',

* local name of remote partner
OSI-LPAP      lcltname,
              ASSOCIATION-NAMES = assoname,
              ASSOCIATIONS = 2, CONTWIN = 1, CONNECT = 1,
              APPLICATION-CONTEXT = {XATMIAC| XATMICCR}

```



**XATMICCR is necessary for distributed transactions.**

```

* local name of connection
OSI-CON       lconname,
              P-SEL=*NONE, S-SEL=*NONE,
              T-SEL=C'tsel', N-SEL=C'machname',
              LOCAL-ACCESS-POINT = server,
              OSI-LPAP = lcltname

```

- The statements CON, LPAP and LSES must be specified for LU6 connections to an openUTM or OpenCPIC partner, see the openUTM manual “Generating Applications”.
- No automatic conversion may be generated, i.e. MAP = USER must be specified in the OSI-CON statement (for OSI TP partners) or in the SESCHA statement (for LU6.1 partners) when generating connections to the partner application.
- A PTERM statement must be specified for each UPIC partner, e.g. :

```

PTERM tnsclient, PTYPE=UPIC-R, PRONAM=DxxxSyyy (with UPIC-Remote)
PTERM client1, PTYPE=UPIC-L (with UPIC-Local)

```

If a UPIC partner is implementing the user concept of openUTM and therefore transfers security data (user ID, password), USER statements must also be issued for each user ID used.

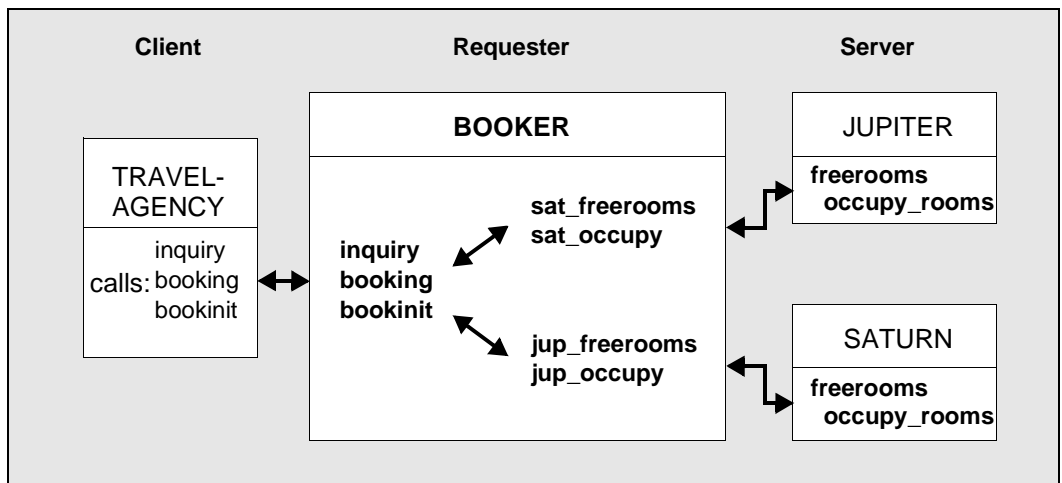
#### 4.5.3.4 Defining the services used

An LTAC statement must be specified for each service called. If `xatmigen` is called with the “utm” carrier system, `xatmigen` automatically creates an LTAC statement of the form `LTAC ltac` for each SVCU statement.

You must still supplement these LTAC statements with the operand `LPAP=` and possibly other operands such as `RTAC=` or `TYPE=`. All other KDCDEF statements for the remote partner, such as `LPAP` or `OSI-LPAP`, must likewise be inserted.

#### 4.5.3.5 Example of requester generation

The example below illustrates the steps involved in XATMI requester generation. The generation is for a central “hotel reservation application” “BOOKER” (requester) which is linked on the one hand to “TRAVEL-AGENCY” client applications in the travel agency and on the other hand to the two hotel management applications “JUPITER” and “SATURN” (servers) in the hotels.



BOOKER offers the services “inquiry”, “booking”, and “bookinit”, which can be called by the clients; each of the two servers offers the end services “freerooms” and “occupy\_rooms”. The following apply:

- “bookinit” is an end service and is used to initialize BOOKER.
- “inquiry” and “booking” are intermediate services which request rooms in the various hotels and then book the room. They are implemented as the functions “inqproc” and “bookproc”.

- “inquiry” uses the conversational paradigm and calls the end services “freeroms” in the two hotels. “freeroms” is addressed in BOOKER by “sat\_freeroms” or “jup\_freeroms”. The “freeroms” service does not require typed buffers.
- “booking” uses the synchronous request/response paradigm and calls the end services “occupy\_rooms” in the two hotels. “occupy\_rooms” is addressed in BOOKER by “sat\_occupy” or “jup\_occupy”. The “occupy-rooms” service requires a personal data record “t\_person” under the buffer name “personbuffer”.

The generation is carried out in the following steps:

1. Supply data structures (LC description files).

The participating partners must agree on uniform data structures and each must make itself known to the others. These data structures (in C or COBOL) describe the typed buffers:

**userbuf.h file**

```
typedef struct {
    char name[20];
    char sex;
    short persons;
    float huge;
    int duration_of_last_visits[16];
} t_person;
```

2. Create the input file for the local configuration file (LC definition file).

The services and buffers used are defined in this file:

**booker.def file**

```
# services provided:
SVCP bookinit;
REQP inquiry, PROG=inqproc, MODE=CV;
REQP booking, TAC=booktac, PROG=bookproc, MODE=RR;

# services called:
SVCU sat_freeroms, RSN=freeroms, TAC=frmtac, DEST=SATURN;
SVCU sat_occupy, RSN=occupy_room, TAC=occtac, DEST=SATURN \
    BUFFERS=(personbuffer);
SVCU jup_freeroms, RSN=freeroms, TAC=frmtac, DEST=JUPITER;
SVCU jup_occupy, RSN=occupy_room, TAC=occtac, DEST=JUPITER \
    BUFFERS=(personbuffer);

# subtypes used:
BUFFER personbuffer, REC=t_person;
```

### 3. Generate the local configuration

The local configuration is created with the `xatmigen` utility program. When called, the input files for the typed buffer and for the LCF must be specified. If the generation run is error-free, `xatmigen` creates the local configuration file, in this example together with the frame for KDCDEF.

X/W

Call on Unix, Linux and Windows systems:

X/W

```
xatmigen utm -c C -l booker.lcf -d booker.def userbuf.h
```

B

Call on BS2000 systems:

B

```
/START-XATMIGEN
```

B

```
% ENTER CCM0001 PARAMETER:
```

B

```
* params utm -c C -l BOOKER.LCF -d BOOKER.DEF USERBUF.H
```

`xatmigen` creates the following files:

**booker.lcf file:**

```
# services provided:
SVCU bookinit TAC=bookinit PROG=bookinit COMP=C MODE=RR;
REQP inquiry TAC=inquiry PROG=inqproc COMP=C MODE=CV;
REQP booking TAC=booktac PROG=bookproc COMP=C MODE=RR;
**COMP = ILCS on BS2000 systems**

# services called:
SVCU sat_freeroms RSN=freeroms TAC=frmtac DEST=SATURN MODE=RR;
SVCU sat_occupy RSN=occupy_room TAC=occtac DEST=SATURN MODE=RR \
    BUFFERS=(personbuffer);
SVCU jup_freeroms RSN=freeroms TAC=frmtac DEST=JUPITER MODE=RR;
SVCU jup_occupy RSN=occupy_room TAC=occtac DEST=JUPITER MODE=RR \
    BUFFERS=(personbuffer);

# subtypes used:
BUFFER personbuffer REC=t_person TYPE=X_C_TYPE LEN=91 \
    SYNTAX=C20csfI16;
```

**xtutm.def file:**

```
*#services provided:
TAC bookinit, PROGRAM = bookinit, API = (XOPEN,XATMI)
PROGRAM bookinit, COMP = C // COMP = ILCS on BS2000 systems
MAX TASKS-IN-PGWT=1
TACCLASS 1, TASKS-FREE=1, PGWT=YES
TAC inquiry, PROGRAM = inqproc, API = (XOPEN,XATMI)
PROGRAM inqproc, COMP = C // COMP = ILCS on BS2000 systems
TAC booktac, PROGRAM = bookproc, API = (XOPEN,XATMI)
PROGRAM bookproc, COMP = C // COMP = ILCS on BS2000 systems
* #services called:
LTAC frmtac , WAITTIME=(10,30)
```

```
LTAC    occtac , WAITTIME=(10,30)
LTAC    frmtac , WAITTIME=(10,30) // delete afterwards
LTAC    occtac , WAITTIME=(10,30) // delete afterwards
*# used buffer
```

#### 4. Generating the UTM application

First of all, the input file of the KDCDEF run must be created (e.g. using an editor). As a template use the “xtutm.def” file, generate the connection to the HOTEL application, and add all other necessary KDCDEF statements; see the openUTM manual “Generating Applications”. A mixture of XATMI program units, CPI-C program units, and KDCS program units is permitted in an UTM application.

Then proceed with the UTM application as with any other UTM application, i.e. generate with KDCDEF, link the application, and start it.



## 4.6 Creating XATMI applications

An XATMI application is an UTM application containing XATMI program units. It is therefore started in the same way as any “normal” UTM application.

Include files for C programs and COPY elements for COBOL programs are available for creating XATMI applications under openUTM.

The library which you need for linking is described in the sections "[Linking the application on Unix, Linux and Windows systems](#)" and "[Linking the application on BS2000 systems](#)".

### 4.6.1 Include files and COPY elements

The following include files are required in C modules with XATMI calls:

- X/W ● On Unix, Linux and Windows systems the include file `xatmi.h` which is located in the following directory:
  - X – `utmpath/xatmi/include` (Unix and Linux systems)
  - W – `utmpath\xatmi\include` (Windows systems)
- X/W `xatmi.h` includes the supplied `xatmidef.h` file, located in the same directory.
- B ● On BS2000 systems, the XATMI.H. include file
  - B This file is contained in the `$userid.SYSLIB.UTM-D.065.XOPEN` library.
  - B (XATMI.H includes the XATMIDEF.H file which is part of delivery and located in the same library).
  - B
- The file(s) with the data structures for all typed buffers used in the module, see also [page 110](#).

The following COPY elements are required for COBOL modules with XATMI calls:

- The TPSTATUS, TPTYPE, TPSVCDEF, TPSVCRET COPY elements and TPRETURN.
- X/W ● On Unix, Linux and Windows systems the include file `xatmi.h` which is located in the following directory:
  - X – `utmpath/xatmi/copy-cobo185` bzw. `utmpath/xatmi/netcobo1` (Unix and Linux systems)
  - X
  - W – `utmpath\xatmi\copy-cobo185` bzw. `utmpath\xatmi\netcobo1` (Windows systems)
- B On BS2000 systems, these COPY elements are in the
  - B `$userid.SYSLIB.UTM.065.XOPEN` library.
- The file(s) with the data structures for all “typed records” used in the module.

**B**  
**B** On BS2000 systems, all program units must be compiled with the LINKAGE option for ILCS.

## 4.6.2 Linking the application on Unix, Linux and Windows systems

**X/W** When you link an XATMI server application, the following libraries must also be linked.

- X/W** ● All service program units and modules
- X** ● On Unix and Linux systems, the XATMI server library *utmpath/sys/libxopen*
- X** ● The OSS library; (required for OSI TP protocol only)
- X/W** ● The openUTM library
- X** ● On Unix and Linux systems, the mathematical library (*-lm* statement)

## 4.6.3 Linking the application on BS2000 systems

**B** When linking an XATMI server application, the following libraries must also be linked.

- B** 1. all service program units and modules
- B** 2. The *\$userid.SYSLIB.UTM.065.XOPEN* XATMI server library
- B** 3. The libraries for CRTE, e.g. *SYSLNK.CRTE*
- B** 4. The openUTM library

## 4.7 Environment or job variables for XATMI

openUTM evaluates a number of environment variables (on Unix, Linux and Windows systems) or job variables (on BS2000 systems) for XATMI applications. The environment variables or job variables must be set before the application is started.

Traces can be activated for error diagnosis during application runtime, see [section "Controlling the trace"](#).

### 4.7.1 Environment variables on Unix, Linux and Windows systems

**X/W** For an XATMI application, the following environment variables are evaluated:

<b>X/W</b>	<b>XTLCF</b>	Name of the local configuration file (LCF)
<b>X/W</b>		The file name of the local configuration file must comply with the operating system conventions.
<b>X/W</b>		If this variable is not set, a search is made under the name <code>xatmlcf</code> in the current directory.
<b>X/W</b>	<b>XTPALCF</b>	Defines the search path for additional descriptions of typed buffers.
<b>X/W</b>		The buffer descriptions are read from local configuration files with the name <code>xatmlcf</code> or from the name specified in <code>XTLCF</code> .
<b>X/W</b>		A search for all important XATMI generations (e.g. SVCU ...) is performed in the local configuration file specified using <code>XTLCF</code> .
<b>X/W</b>		A search for local configuration files is performed in all the directories specified in <code>XTPALCF</code> and the typed buffer descriptions are gathered internally (If multiple buffers have the same name only the first buffer description is used).
<b>X/W</b>		The search path structure is exactly the same as in the <code>PATH</code> environment variable:
<b>X</b>		<i>directory1:directory2: ...</i> (Unix and Linux systems)
<b>W</b>		<i>directory1;directory2; ...</i> (Windows systems)

## 4.7.2 Setting job variables on BS2000 systems

B		For an XATMI application, you can set job variables which are linked with the application via the following names (link names):
B		
B	XTLCF	Link to job variable containing the file name of the local configuration file (LCF).
B		The file name of the local configuration file must comply with the operating system conventions.
B		A search is performed for the file in the current user ID.
B		If XTLCF is not allocated to a job variable, a search is performed under the name XATMILCF in the current user ID.
B		
B	XTPALCF	Link to job variable containing the search path for additional descriptions of typed buffers.
B		The buffer description are read from local configuration files with the name XATMILCF or from the name specified in XTLCF.
B		A search for all important XATMI generations (e.g. SVCU ...) is performed in the local configuration file specified via XTLCF.
B		
B		A search for local configuration files is performed in all the user IDs specified in the search path and the typed buffer descriptions are gathered internally from these files (If multiple buffers have the same name, only the first buffer description is used).
B		
B		The search path is specified in the format <i>id1:id2: ....</i>

## 4.8 Error diagnostics in XATMI programs

XATMI calls are mapped to KDCS calls. The UTM dump therefore also contains only the KDCS calls in the case of XATMI program units. However, for the purpose of diagnosing XATMI programs, you can also generate a trace of the XATMI calls in addition to the UTM dump.

### 4.8.1 Controlling the trace

The XATMI trace can be controlled as follows:

- The UTM start parameter XATMI-TRACE can be used to enable the trace when the application is started, see the relevant openUTM manual “Using UTM Applications”.
- The trace can be enabled or disabled during operation using WinAdmin or WebAdmin. To do this go to the dialog *Properties of UTM Application*, tab *Diagnosis and Account*, field *XATMI Trace*.
- The trace can be enabled or disabled during operation via the KDCADMI administration program interface. This is done using the field *xatmi\_trace* in the data structure *kc\_diag\_and\_account\_par\_str*, see openUTM manual “Administering Applications”.

You can set the following trace levels:

Level	Meaning
ERROR	Only errors are logged.
INTERFACE	Includes the ERROR level. XATMI calls are also logged.
FULL	Includes the INTERFACE level. All KDCS calls to which the XATMI calls are mapped are also logged.
DEBUG	Includes the FULL level. Diagnostic information is also logged.
OFF	Trace is disabled.

## 4.8.2 Name of the trace file

### X/W Trace file on Unix, Linux and Windows systems

X/W The trace records are written to a file in the application directory (*filebase*).

X/W – For stand-alone applications, this file has the name `KDC.TRC.XATMI.appliname.pid`.

X/W – For UTM cluster applications, this file has the name

X/W `KDC.TRC.XATMI.appliname.nodename.pid`.

X/W Where:

X/W appliname

X/W Name of the application.

X/W nodename

X/W Name of the cluster node on which the node application is running.

X/W pid PID of the process.

### B Trace file on BS2000 systems

B By default, the trace records are written to a file.

B – For stand-alone applications, this file has the name `KDC.TRC.XATMI.appliname.tsn`.

B – For UTM cluster applications, this file has the name

B `KDC.TRC.XATMI.appliname.nodename.tsn`.

B Where:

B appliname

B Name of the application.

B nodename

B Name of the cluster node on which the node application is running.

B tsn TSN of the UTM task

B In the UTM start procedure, you can also set up a different trace file for each task and use the `SET-FILE-LINK` command to assign it the link name `KDCXATMI`.

## 4.9 Interaction with the TX interface

For a description of the TX interface under openUTM please see chapter 5.

Under openUTM, XATMI programs are always servers. XATMI programs under openUTM contain no TX calls since only XATMI client programs require TX calls for transaction control.

When an XATMI service is called, the client uses the call parameter flag (in C) or the TPTRAN-FLAG (in COBOL) to control whether or not a called UTM service is included in the global transaction.

The XATMI-C interface includes the service in the global transaction by default. In order to exclude the service from the global transaction, you must set the TPNOTRAN flag explicitly. No default value exists for the XATMI-COBOL interface, you must set either TPTRAN or TPNOTRAN.

If the service is started with the TPTRAN flag, it is included in the global transaction and TX calls are not required.

When using the `tpreturn()` call, the parameter *rval* returns the values TPSUCCESS or TPFAIL. This determines whether the transaction is terminated successfully or rolled back.

For further information on the interaction between the TX and XATMI interfaces refer to the X/OPEN specification "Distributed Transaction Processing: The XATMI Specification", section "Transaction Functions Affecting the XATMI Interface".

## 4.10 Messages

The messages for XATMI and `xatmigen` are listed below.

### XATMI messages

XATMI messages have the form `XTnn messagetext...` and are output to `stderr` on Unix, Linux and Windows systems or `SYSOUT` on BS2000 systems.

XT01 KATMI(&PID) SERVER initiated  
LC-file : &LCF  
Tracefile: &TRACEFILE

#### Meaning

Start message:

&PID Process ID (on Unix, Linux and Windows systems) or task number (on BS2000s systems)

&LCF Name of the local configuration file

&TRACEFILE Name of trace file without generation number

XT02 KATMI(&PID) Local configuration: &ERRTXT

#### Meaning

Error in the local configuration file. &ERRTXT returns a supplementary text.

XT03 XATMI(&PID) Error: &ERRTXT

#### Meaning

Error when calling an XATMI function. &ERRTXT returns a supplementary text.

XT04 KATMI(&PID) System error: &ERRTXT

#### Meaning

System error

### xatmigen messages

`xatmigen` messages have the form `XGnn messagetext...` and are output to `stderr` on Unix, Linux and Windows systems or `SYSOUT` on BS2000 systems.

X On Unix and Linux systems, use the `LANG` environment variable to control whether you want German or English messages.

B On BS2000 systems you can assign the link name `LANG` to different job variables on a task-specific basis and set the value 'D' or 'E' as the language identifier. In this way, you can specify whether you want to receive German or English messages.



XG01            Generation of the local configuration files: &LCF / &DEF / &CODE

**Meaning**

Start message of Tool.

&LCF            name of local configuration file created

&DEF            name of generation fragment created

&CODE          string code for character array

XG02            Generation terminated successfully

**Meaning**

The LCF was created; generation was terminated successfully.

XG03            Generation terminated successfully with warnings

**Meaning**

The LCF was created. Nevertheless, a warning is output because unnecessary files were specified, for example. However, this warning has no effect on the generation.

XG04            Generation terminated by error

No file created.

**Meaning**

The LCF was not created; the generation could not be performed. The cause can be determined from previous messages

XG05            &FTYPE file'&FNAME'

**Meaning**

This message specifies the file currently being edited, in the following form:

&FTYPE:        "description" file contains data structures

                 "definition" file contains the LCF input

                 "LC" file contains the local configuration

&FNAME:        Filename

XG10            Call: &PARAM

**Meaning**

Syntax error when calling XATMIGEN:

PARAM:        possible call parameters and switches

XG11            [Error] Cannot create &FTYPE file 'FNAME

&REASON

**Meaning**

The &FNAME file of type &FTYPE cannot be created

&REASON contains a more precise explanation.

&FTYPE:        GEN = generation fragment file (=generation statements)

                 LC = local configuration file

XG12 [Warning] File not found.

**Meaning**

The definition file or a description file was not found; perhaps the file does not exist.

XG13 [Warning] Too many &OBJECTS, Maximum: &MAXNUM

**Meaning**

Message indicating that too many objects were found.

&OBJECTS: subtypes

&MAXNUM: maximum number

XG14 [Error] Line &LINE: Syntaxerror, &helptext

**Meaning**

Syntax error in line &LINE of the LC definition file

&HELPTXT: help text

XG15 [Error] Line &LINE: No record definition found for buffer &BUFF

**Meaning**

No associated record definition could be found for the buffer &BUFF in line &LINE.

XG16 [Error] Line &LINE: Basicstype error in buffer &BUFF

**Meaning**

The syntax description of the buffer &BUFF in line &LINE of the LCF contains an incorrect basic type (int, short, etc.).

XG17 [Error] Cannot open &FTYPE file '&FNAME'.  
&REASON

**Meaning**

The &FNAME file of type &FTYPE cannot be opened..

&REASON: contains a more detailed explanation.

&FTYPE: DEF (= LC definition file)

XG18 [Error] &REASON

**Meaning**

General error.

&REASON contains a detailed reason for the error.

XG19 [Message] Created new buffer: '&BUFF'

**Meaning**

&BUFF: created buffer

XG20 [Message] Service name '&SVC' truncated to 16 characters!

**Meaning**

&SVC : service name.

XG21 [Message] Line &LINE: unknown statement line '&HELPTTEXT'

**Meaning**

Message for the line &LINE in the LC definition file  
&HELPTTEXT: help text (part of LC-line)

XG22 [Message] Line &LINE: Default set MODE='&TEXT'

**Meaning**

Message for the line &LINE in the LC definition file  
&TEXT: set default mode

## 4.11 T.61 character set

	0	1	2	3	4	5	6	7	8	9	...	F
0			SP	0	@	P		p				
1			!	1	A	Q	a	q				
2			"	2	B	R	b	r				
3			#	3	C	S	c	s				
4			¤	4	D	T	d	t				
5			%	5	E	U	e	u				
6			&	6	F	V	f	v				
7			'	7	G	W	g	w				
8	BS		(	8	H	X	h	x				
9		SS2	)	9	I	Y	i	y				
A	LF	SUB	*	:	J	Z	j	z				
B		ESC	+	;	K	[	k		PLD	CSI		
C	FF		,	<	L		l		PLU			
D	CR	SS3	-	=	M	]	m					
E	LS1		.	>	N		n					
F	LS0		/	?	O	-	o					

Code table T.61 in accordance with CCITT recommendation

Meaning of abbreviations:

BS= BACKSPACE

SUB= SUBSTITUTE CHARACTER

LF= LINE FEED

ESC= ESCAPE

FF= FORM FEED

SS3= SINGLE-SHIFT THREE

CR= CARRIAGE RETURN

SP= SPACE

LS1= LOCKING SHIFT ONE

PLD= PARTIAL LINE DOWN

LS0= LOCKING SHIFT ZERO

PLU= PARTIAL LINE UP

SS2= SINGLE-SHIFT TWO

CSI= CONTROL SEQUENCE INTRODUCER

---

## 5 X/Open TX interface

TX (Transaction Demarcation Interface) is a standardized X/Open program interface for defining transactions across hosts. It is available in both COBOL and C.

This chapter describes the particularities of the TX interface under openUTM. For a detailed description of the TX interface and the call formats, refer to the X/Open CAE specification “Distributed Transaction Processing: The TX (Transaction Demarcation) Specification” of April 1995.

Below it is assumed that readers are familiar with this specification.

### 5.1 *transaction\_control* characteristic

The TX interface allows you to execute transactions either chained or unchained.

If the *transaction\_control* characteristic has the value TX\_CHAINED, then you need only start the first transaction explicitly: the end of transaction implicitly marks the start of the next transaction.

If the *transaction\_control* characteristic has the value TX\_UNCHAINED, then you have to mark the start of each transaction explicitly.

openUTM always works with *transaction\_control*=TX\_CHAINED.

When a service starts under openUTM a transaction is automatically initiated. This means that it is also unnecessary to mark the first transaction.

## 5.2 TX interface calls under openUTM

The table below lists all the TX calls which are available under openUTM.

C call	COBOL call	Description
tx_commit	TXCOMMIT	terminate global transaction successfully
tx_rollback	TXROLLBACK	roll back global transaction
tx_info	TXINFORM	query global transaction information
tx_set_commit_return	TXSETCOMMITRET	set <i>commit_return</i> characteristic
tx_set_transaction_control	TXSETTRANCTL	set <i>transaction_control</i> characteristic
tx_set_transaction_timeout	TXSETTIMEOUT	set <i>transaction_timeout</i> characteristic
tx_open	TXOPEN	open set of Resource Managers

### 5.2.1 Particularities of TX calls under openUTM

*tx\_commit, tx\_rollback*

If reception of the value CM\_TAKE\_COMMIT\_DEALLOCATE in *status\_received* results in a tx\_commit or tx\_rollback call, then control is not returned to the CPI-C program unit. tx\_commit() or tx\_rollback() is internally mapped to the KDCS PEND call and the server program is terminated.

The tx\_commit and tx\_rollback calls are not permitted if one or more of the partners involved in the global transaction are communicating via the LU6.1 protocol.

*tx\_set\_commit\_return*

Under openUTM the only value permitted is TX\_COMMIT\_COMPLETED.

The value TX\_COMMIT\_DECISION\_LOGGED is rejected with TX\_NOT\_SUPPORTED.

*tx\_set\_transaction\_control*

Under openUTM the only value permitted is TX\_CHAINED.

The value TX\_UNCHAINED is rejected with TX\_PROTOCOL\_ERROR.

*tx\_open*

Always returns the value TX\_OK.

### Unsupported calls

*tx\_begin*

This call is rejected with TX\_PROTOCOL\_ERROR since openUTM automatically opens a transaction when a service is started.

*tx\_close*

This call is rejected with TX\_PROTOCOL\_ERROR since it is only permitted when no transaction is being processed. However, a transaction is always open under openUTM.

## 5.3 Interaction with the CPI-C interface

In CPI-C program units, the conversation characteristic *Sync\_Level* controls the integration of a service into a global transaction. If *Sync\_Level* has the value CM\_SYNC\_POINT or CM\_SYNC\_POINT\_NO\_CONFIRM the service is integrated into the global transaction.

If the CPI-C interface is used, the end of transaction is always requested by the client.

Services are not restarted at the CPI-C interface following a malfunction or system crash as they are at the KDCS interface.

Under openUTM it is necessary to distinguish between two Cases of TX call functionality in CPI-C services:

### Case 1: the initiator integrates the acceptor into a global transaction

If a client (initiator) integrates a CPI-C service which is running under openUTM into a global transaction then the TX calls in a CPI-C service running under openUTM are used for confirmation purposes.

In this case, the acceptor (server) receives the value CM\_SYNC\_POINT or CM\_SYNC\_POINT\_NO\_CONFIRM in the Extract\_Sync\_Level call. In the event of a Receive call, the value passed by the client to the *status\_received* field determines whether a tx\_commit() or tx\_rollback() is required in the called service.

Case 1 occurs if the client

- is an UTM application which has established the connection via the OSI TP protocol and has selected the COMMIT *Functional Unit*
- is an OpenCPIC application which has established the conversation with *Sync\_Level* CM\_SYNC\_POINT or CM\_SYNC\_POINT\_NO\_CONFIRM
- is an external application which is using the CPI-C interface and has established the conversation with *Sync\_Level* CM\_SYNC\_POINT or CM\_SYNC\_POINT\_NO\_CONFIRM

### **Case 2: the initiator does not integrate the acceptor into a global transaction**

If a client (initiator) does not integrate a CPI-C service which is running under openUTM into a global transaction then the called server program implicitly becomes the root of a global transaction. The TX calls then function as requests and control the global transaction.

In this case, the acceptor (server) receives the value CM\_NONE or CM\_CONFIRM in the Extract\_Sync\_Level call. In the server program, the tx\_commit() and tx\_rollback() calls determine whether the global transaction is successfully concluded or rolled back.

Case 2 occurs if the client

- is an UTM application which has established the connection via the OSI TP protocol and has not selected the COMMIT *Functional Unit*
- is an OpenCPIC client which has established the conversation with *Sync\_Level* CM\_NONE or CM\_CONFIRM
- is a UPIC client
- is an external application which is using the CPI-C interface and has established the conversation with *Sync\_Level* CM\_NONE or CM\_CONFIRM

For further information on interaction between the TX and CPI-C interfaces, refer to the X/Open specification "Distributed Transaction Processing: The CPI-C Specification, Version 2", section "Effects of Calls on Half-Duplex Conversations to X/Open TX Interface".

The chapter entitled "Program-to-Program Communication Tutorial" in the X/Open specification also contains two example scenarios for the use of TX calls in CPI-C programs. These examples correspond to the processing sequence under openUTM: "Sending Program Issues a Commit" and "Two Chained Transactions".

In these examples, openUTM is always "System Y", that is to say the server.

The functionality of the TX interface in UTM client programs is described in the manual "openUTM-Client for the OpenCPIC Carrier System".



## 5.4 Interaction with the XATMI interface

TX calls are not needed for transaction control in XATMI services under openUTM.

The call parameter *flag* (in C) or the TPTRAN-FLAG field (in COBOL) determine whether or not XATMI services are integrated into global transactions.

The XATMI call `tpreturn()` determines whether a transaction is successfully terminated or rolled back. No TX calls are required.

The transaction is successfully terminated or rolled back depending on whether the value TPSUCCESS or TPFALL is returned in the *rval* parameter.

The service is integrated into the global transaction by default at the XATMI-C interface. If you do not want to integrate the service into the global transaction, you must specify this explicitly by setting the TPNOTRAN flag.

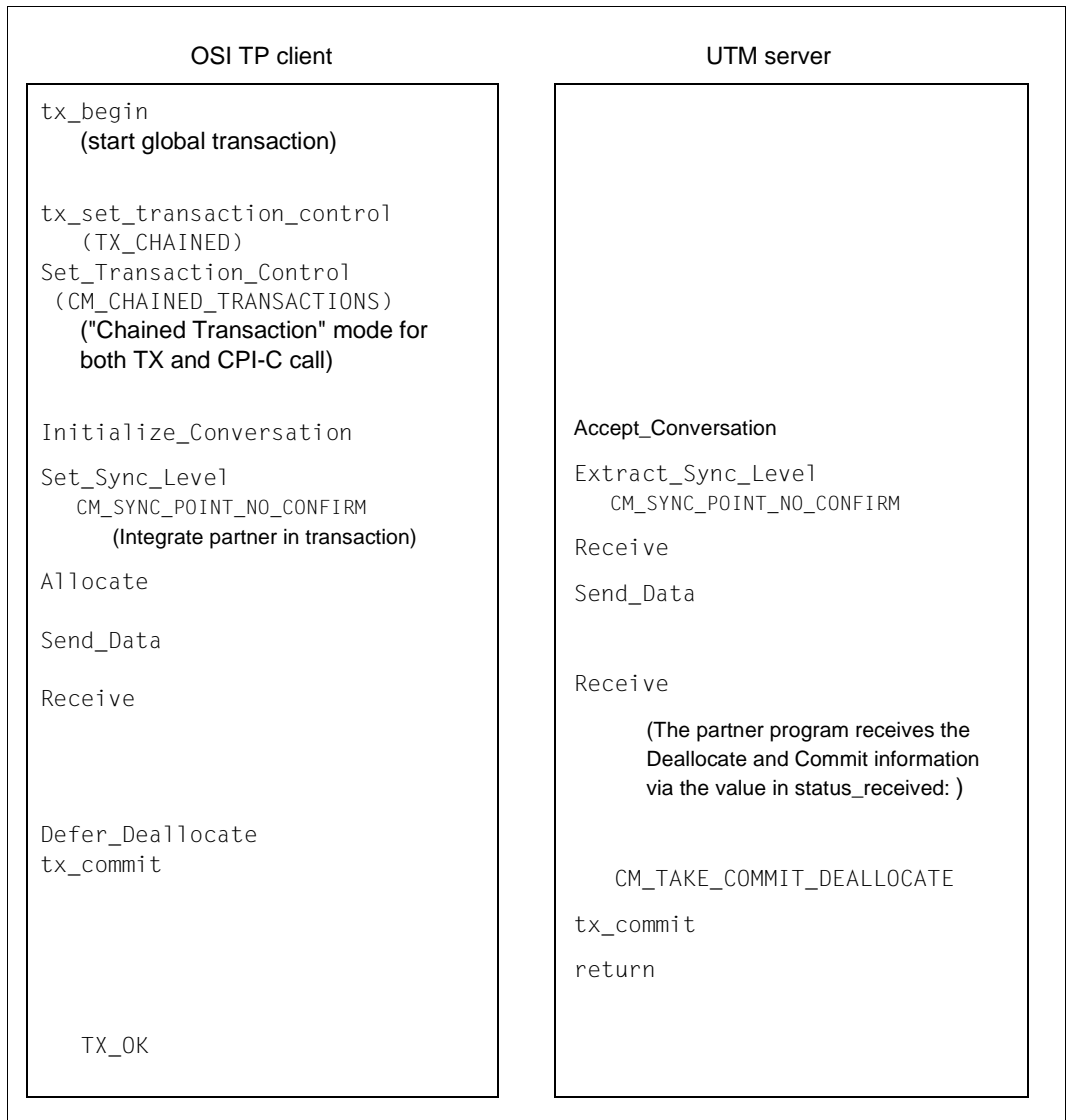
There is no default value for the XATMI-COBOL interface and you must set either TPTRAN or TPNOTRAN.

For further information on the interaction between the TX and XATMI interfaces, refer to the X/Open specification "Distributed Transaction Processing: The XATMI Specification", section "Transaction Functions Affecting the XATMI Interface".

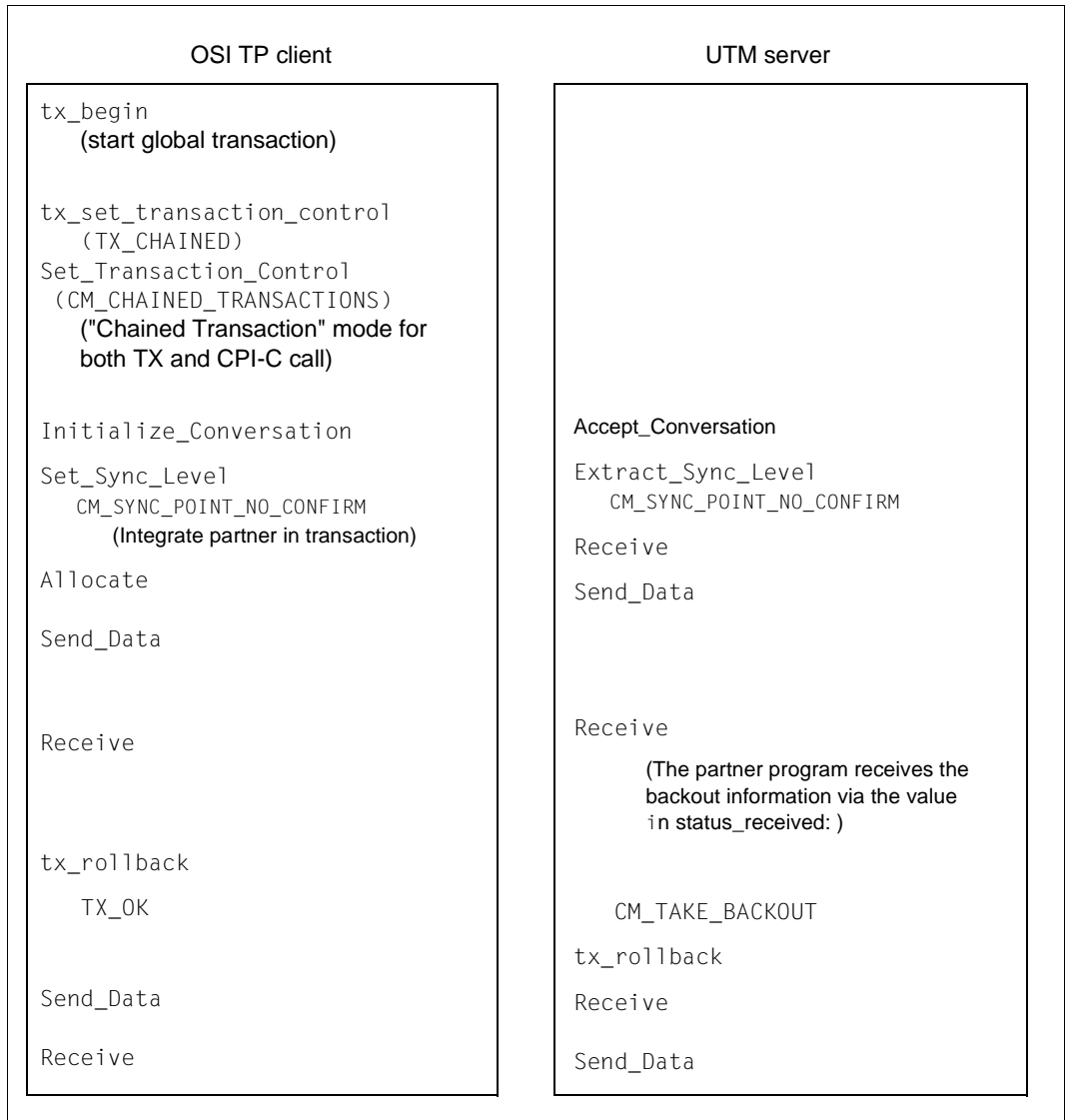
## 5.5 Examples for the use of the TX interface

The two flow charts below indicate how the CPI-C and TX interfaces have to be coordinated if the client is to integrate the called service into a global transaction.

1. Server program is integrated into the global transaction and the transaction is terminated successfully



2. The server program is integrated into the global transaction and the transaction is rolled back



**Note:**

In the case of multi-step transactions, you should query the status of the conversation with which you want to maintain communication after every TX transaction control call (`Extract_Conversation_State`). The returned status informs you whether the conversation is in *Send* or *Receive* state. You can then continue the program run with a Receive or Send call as appropriate.

This query prevents `CM_STATE_CHECK` errors which can occur if the sequence of Send and Receive calls is incorrect.

## 5.6 Creating an application with TX calls

An include file is supplied with openUTM to support you when creating programs containing TX calls in C. COPY elements are supplied for COBOL.

### X/W Files and libraries for the TX interface on Unix, Linux and Windows systems

X/W ● The include file `tx.h` is supplied for C. This file is located under:

X – `utmpath/tx/include/tx.h` (Unix and Linux systems)

W – `utmpath\tx\include\tx.h` (Windows systems)


X/W ● The COPY elements `TXSTATUS` and `TXINFDEF` are supplied for COBOL. These files are located under:

X – `utmpath/tx/copy-cobol85` or `utmpath/tx/netcobol` (Unix and Linux systems)

W – `utmpath\tx\copy-cobol85` or `utmpath\tx\netcobol` (Windows systems)

X ● You must use the following library to link the programs on Unix and Linux systems

X – `utmpath/sys/libxopen`

W  On Windows systems, the X/Open functions are included in the UTM library `libwork.lib`.

### B Files and libraries for the TX interface on BS2000 systems

B The library `$userid.SYSLIB.UTM-D.065.XOPEN` is supplied on BS2000 systems.

B You must use this library to link the programs.

B The include file for C (`TX.H`) and the COPY elements for COBOL (`TXSTATUS` and `TXINFDEF`) are also present in this library as type S elements.

## Generating an application with TX calls

No special generation is necessary when you use the TX interface.

Under openUTM, TX calls are only of use in CPI-C programs. To generate CPI-C applications, refer to [section “Generating a CPI-C application” on page 97](#).

The TX library does not issue any messages.

## 5.7 Diagnosing errors in TX calls

TX calls are mapped to KDCS calls. Consequently, the UTM dump only contains KDCS calls. However, you can generate a TX call trace in addition to the UTM dump in order to diagnose errors which occur during transaction control.

### 5.7.1 Controlling the trace

The TX trace can be controlled as follows:

- The UTM start parameter TX-TRACE can be used to enable the trace when the application is started, see the relevant openUTM manual “Using UTM Applications”.
- The trace can be enabled or disabled during operation using WinAdmin or WebAdmin. To do this go to the dialog *Properties of UTM Application*, tab *Diagnosis and Account*, field *TX Trace*.
- The trace can be enabled or disabled during operation via the KDCADMI administration program interface. This is done using the field *tx\_trace* in the data structure *kc\_diag\_and\_account\_par\_str*, see openUTM manual “Administering Applications”.

You can set the following trace levels:

Level	Meaning
ERROR	Only errors are logged.
INTERFACE	Includes the ERROR level. TX calls are also logged.
FULL	Includes the INTERFACE level. All KDCS calls to which the TX calls are mapped are also logged.
DEBUG	Includes the FULL level. Diagnostic information is also logged.
OFF	Trace is disabled.

## 5.7.2 Name of the trace file

### X/W Trace file on Unix, Linux and Windows systems

X/W The trace records are written to a file in the application directory (*filebase*).

X/W – For stand-alone applications, this file has the name `KDC.TRC.TX.appliname.pid`.

X/W – For UTM cluster applications, this file has the name

X/W `KDC.TRC.TX.appliname.nodename.pid`.

X/W Where:

X/W appliname

X/W Name of the application.

X/W nodename

X/W Name of the cluster node on which the node application is running.

X/W pid PID of the process.

### B Trace file on BS2000 systems

B By default, the trace records are written to a file.

B – For stand-alone applications, this file has the name `KDC.TRC.TX.appliname.tsn`.

B – For UTM cluster applications, this file has the name

B `KDC.TRC.TX.appliname.nodename.tsn`.

B Where:

B appliname

B Name of the application.

B nodename

B Name of the cluster node on which the node application is running.

B tsn TSN of the UTM task

B In the UTM start procedure, you can also set up a different trace file for each task and use

B the SET-FILE-LINK command to assign it the link name `KDCTX`.

---

# Glossary

A term in *italic* font means that it is explained somewhere else in the glossary.

## **abnormal termination of a UTM application**

Termination of a *UTM application*, where the *KDCFILE* is not updated. Abnormal termination is caused by a serious error, such as a crashed computer or an error in the system software. If you then restart the application, openUTM carries out a *warm start*.

## **abstract syntax (OSI)**

Abstract syntax is defined as the set of formally described data types which can be exchanged between applications via *OSI TP*. Abstract syntax is independent of the hardware and programming language used.

## **acceptor (CPI-C)**

The communication partners in a *conversation* are referred to as the *initiator* and the acceptor. The acceptor accepts the conversation initiated by the initiator with `Accept_Conversation`.

## **access list**

An access list defines the authorization for access to a particular *service*, *TAC queue* or *USER queue*. An access list is defined as a *key set* and contains one or more *key codes*, each of which represent a role in the application. Users or LTERMs or (OSI) LPAPs can only access the service or *TAC queue/USER queue* when the corresponding roles have been assigned to them (i.e. when their *key set* and the access list contain at least one common *key code*).

## **access point (OSI)**

See *service access point*.

## **ACID properties**

Acronym for the fundamental properties of *transactions*: atomicity, consistency, isolation and durability.

## **administration**

Administration and control of a *UTM application* by an *administrator* or an *administration program*.

### **administration command**

Commands used by the *administrator* of a *UTM application* to carry out administration functions for this application. The administration commands are implemented in the form of *transaction codes*.

### **administration journal**

See *cluster administration journal*.

### **administration program**

*Program unit* containing calls to the *program interface for administration*. This can be either the standard administration program *KDCADM* that is supplied with openUTM or a program written by the user.

### **administrator**

User who possesses administration authorization.

### **AES**

AES (Advanced Encryption Standard) is the current symmetric encryption standard defined by the National Institute of Standards and Technology (NIST) and based on the Rijndael algorithm developed at the University of Leuven (Belgium). If the AES method is used, the UPIC client generates an AES key for each session.

### **Apache Axis**

Apache Axis (Apache eXtensible Interaction System) is a SOAP engine for the design of Web services and client applications. There are implementations in C++ and Java.

### **Apache Tomcat**

Apache Tomcat provides an environment for the execution of Java code on Web servers. It was developed as part of the Apache Software Foundation's Jakarta project. It consists of a servlet container written in Java which can use the JSP Jasper compiler to convert JavaServer pages into servlets and run them. It also provides a fully featured HTTP server.

### **application cold start**

See *cold start*.

### **application context (OSI)**

The application context is the set of rules designed to govern communication between two applications. This includes, for instance, abstract syntaxes and any assigned transfer syntaxes.



**application entity (OSI)**

An application entity (AE) represents all the aspects of a real application which are relevant to communications. An application entity is identified by a globally unique name (“globally” is used here in its literal sense, i.e. worldwide), the *application entity title* (AET). Every application entity represents precisely one *application process*. One application process can encompass several application entities.

**application entity qualifier (OSI)**

Component of the *application entity title*. The application entity qualifier identifies a *service access point* within an application. The structure of an application entity qualifier can vary. openUTM supports the type “number”.

**application entity title (OSI)**

An application entity title is a globally unique name for an *application entity* (“globally” is used here in its literal sense, i.e. worldwide). It is made up of the *application process title* of the relevant *application process* and the *application entity qualifier*.

**application information**

This is the entire set of data used by the *UTM application*. The information comprises memory areas and messages of the UTM application including the data currently shown on the screen. If operation of the UTM application is coordinated with a database system, the data stored in the database also forms part of the application information.

**application process (OSI)**

The application process represents an application in the *OSI reference model*. It is uniquely identified globally by the *application process title*.

**application process title (OSI)**

According to the OSI standard, the application process title (APT) is used for the unique identification of applications on a global (i.e. worldwide) basis. The structure of an application process title can vary. openUTM supports the type *Object Identifier*.

**application program**

An application program is the core component of a *UTM application*. It comprises the main routine *KDCROOT* and any *program units* and processes all jobs sent to a *UTM application*.

**application restart**

see *warm start*

### **application service element (OSI)**

An application service element (ASE) represents a functional group of the application layer (layer 7) of the *OSI reference model*.

### **application warm start**

see *warm start*.

### **association (OSI)**

An association is a communication relationship between two application entities. The term “association” corresponds to the term *session* in *LU6.1*.

### **asynchronous conversation**

CPI-C conversation where only the *initiator* is permitted to send. An asynchronous transaction code for the *acceptor* must have been generated in the *UTM application*.

### **asynchronous job**

*Job* carried out by the job submitter at a later time. openUTM includes *message queuing* functions for processing asynchronous jobs (see *UTM-controlled queue* and *service-controlled queue*). An asynchronous job is described by the *asynchronous message*, the recipient and, where applicable, the required execution time.

If the recipient is a terminal, a printer or a transport system application, the asynchronous job is a *queued output job*. If the recipient is an *asynchronous service* of the same application or a remote application, the job is a *background job*. Asynchronous jobs can be *time-driven jobs* or can be integrated in a *job complex*.

### **asynchronous message**

Asynchronous messages are messages directed to a *message queue*. They are stored temporarily by the local *UTM application* and then further processed regardless of the job submitter. Distinctions are drawn between the following types of asynchronous messages, depending on the recipient:

- In the case of asynchronous messages to a *UTM-controlled queue*, all further processing is controlled by openUTM. This type includes messages that start a local or remote *asynchronous service* (see also *background job*) and messages sent for output on a terminal, a printer or a transport system application (see also *queued output job*).
- In the case of asynchronous messages to a *service-controlled queue*, further processing is controlled by a *service* of the application. This type includes messages to a *TAC queue*, messages to a *USER queue* and messages to a *temporary queue*. The USER queue and the temporary queue must belong to the local application, whereas the TAC queue can be in both the local application and the remote application.

**asynchronous program**

*Program unit started by a background job.*

**asynchronous service (KDCS)**

*Service which processes a background job. Processing is carried out independently of the job submitter. An asynchronous service can comprise one or more program units/transactions. It is started via an asynchronous transaction code.*

**audit (BS2000 systems)**

*During execution of a UTM application, UTM events which are of relevance in terms of security can be logged by SAT for auditing purposes.*

**authentication**

*See system access control.*

**authorization**

*See data access control.*

**Axis**

*See Apache Axis.*

**background job**

*Background jobs are asynchronous jobs destined for an asynchronous service of the current application or of a remote application. Background jobs are particularly suitable for time-intensive processing or processing which is not time-critical and where the results do not directly influence the current dialog.*

**basic format**

*Format in which terminal users can make all entries required to start a service.*

**basic job**

*Asynchronous job in a job complex.*

**browsing asynchronous messages**

*A service sequentially reads the asynchronous messages in a service-controlled queue. The messages are not locked while they are being read and they remain in the queue after they have been read. This means that they can be read simultaneously by different services.*

**bypass mode (BS2000 systems)**

*Operating mode of a printer connected locally to a terminal. In bypass mode, any asynchronous message sent to the printer is sent to the terminal and then redirected to the printer by the terminal without being displayed on screen.*

### cache

Used for buffering application data for all the processes of a *UTM application*. The cache is used to optimize access to the *page pool* and, in the case of UTM cluster applications, the *cluster page pool*.

### CCS name (BS2000 systems)

See *coded character set name*.

### client

Clients of a *UTM application* can be:

- terminals
- UPIC client programs
- transport system applications (e.g. DCAM, PDN, CMX, socket applications or UTM applications which have been generated as *transport system applications*).

Clients are connected to the UTM application via LTERM partners.

Note: UTM clients which use the OpenCPIC carrier system are treated just like *OSI TP partners*.

### client side of a conversation

This term has been superseded by *initiator*.

### cluster

A number of computers connected over a fast network and which in many cases can be seen as a single computer externally. The objective of clustering is generally to increase the computing capacity or availability in comparison with a single computer.

### cluster administration journal

The cluster administration journal consists of:

- two log files with the extensions JRN1 and JRN2 for global administration actions,
- the JKAA file which contains a copy of the KDCS Application Area (KAA). Administrative changes that are no longer present in the two log files are taken over from this copy.

The administration journal files serve to pass on to the other node applications those administrative actions that are to apply throughout the cluster to all node applications in a UTM cluster application.

### cluster configuration file

File containing the central configuration data of a *UTM cluster application*. The cluster configuration file is created using the UTM generation tool *KDCDEF*.

**cluster filebase**

Filename prefix or directory name for the *UTM cluster files*.

**cluster GSSB file**

File used to administer GSSBs in a *UTM cluster application*. The cluster GSSB file is created using the UTM generation tool *KDCDEF*.

**cluster lock file**

File in a *UTM cluster application* used to manage cross-node locks of user data areas.

**cluster page pool**

The cluster page pool consists of an administration file and up to 10 files containing a *UTM cluster application's* user data that is available globally in the cluster (service data including LSSB, GSSB and ULS). The cluster page pool is created using the UTM generation tool *KDCDEF*.

**cluster start serialization file**

Lock file used to serialize the start-up of individual node applications (only on Unix systems and Windows systems).

**cluster ULS file**

File used to administer the ULS areas of a *UTM cluster application*. The cluster ULS file is created using the UTM generation tool *KDCDEF*.

**cluster user file**

File containing the user management data of a *UTM cluster application*. The cluster user file is created using the UTM generation tool *KDCDEF*.

**coded character set name (BS2000 systems)**

If the product *XHCS* (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 (*static configuration*) and can be changed dynamically by the administrator (while the application is running, *dynamic configuration*). The configuration is stored in the *KDCFILE*.

### **confirmation job**

Component of a *job complex* where the confirmation job is assigned to the *basic job*. There are positive and negative confirmation jobs. If the *basic job* returns a positive result, the positive confirmation job is activated, otherwise, the negative confirmation job is activated.

### **connection bundle**

see *LTERM bundle*.

**connection user ID**

User ID under which a *TS application* or a *UPIC client* is signed on at the *UTM application* directly after the connection has been established. The following applies, depending on the client (= LTERM partner) generation:

- The connection user ID is the same as the USER in the LTERM statement (explicit connection user ID). An explicit connection user ID must be generated with a USER statement and cannot be used as a “genuine” *user ID*.
- The connection user ID is the same as the LTERM partner (implicit connection user ID) if no USER was specified in the LTERM statement or if an LTERM pool has been generated.

In a *UTM cluster application*, the service belonging to a connection user ID (RESTART=YES in LTERM or USER) is bound to the connection and is therefore local to the node.

A connection user ID generated with RESTART=YES can have a separate service in each *node application*.

**contention loser**

Every connection between two partners is managed by one of the partners. The partner that manages the connection is known as the *contention winner*. The other partner is the contention loser.

**contention winner**

A connection's contention winner is responsible for managing the connection. Jobs can be started by the contention winner or by the *contention loser*. If a conflict occurs, i.e. if both partners in the communication want to start a job at the same time, then the job stemming from the contention winner uses the connection.

**conversation**

In CPI-C, communication between two CPI-C application programs is referred to as a conversation. The communication partners in a conversation are referred to as the *initiator* and the *acceptor*.

**conversation ID**

CPI-C assigns a local conversation ID to each *conversation*, i.e. the *initiator* and *acceptor* each have their own conversation ID. The conversation ID uniquely assigns each CPI-C call in a program to a conversation.

### **CPI-C**

CPI-C (Common Programming Interface for Communication) is a program interface for program-to-program communication in open networks standardized by X/Open and CIW (**CPI-C Implementor's Workshop**). 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<sup>®</sup> MSCF).

### **data access control**

In data access control openUTM checks whether the communication partner is authorized to access a particular object belonging to the application. The access rights are defined as part of the configuration.

### **data space (BS2000 systems)**

Virtual address space of BS2000 which can be employed in its entirety by the user. Only data and programs stored as data can be addressed in a data space; no program code can be executed.

### **dead letter queue**

The dead letter queue is a TAC queue which has the fixed name KDCDLETQ. It is always available to save queued messages sent to transaction codes, TAC queues, LPAP or OSI-LPAP partners 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, LPAP or OSI-LPAP statement's DEAD-LETTER-Q parameter.

### **DES**

DES (Data Encryption Standard) is an international standard for encrypting data. One key is used in this method for encoding and decoding. If the DES method is used, the UPIC client generates a DES key for each session.

### **dialog conversation**

CPI-C conversation in which both the *initiator* and the *acceptor* are permitted to send. A dialog transaction code for the *acceptor* must have been generated in the *UTM application*.



**dialog job, interactive job**

Job which starts a *dialog service*. The job can be issued by a *client* or, when two servers communicate with each other (*server-server communication*), by a different application.

**dialog message**

A message which requires a response or which is itself a response to a request. The request and the response both take place within a single service. The request and reply together form a dialog step.

**dialog program**

*Program unit* which partially or completely processes a *dialog step*.

**dialog service**

*Service* which processes a *job* interactively (synchronously) in conjunction with the job submitter (*client* or another server application) . A dialog service processes *dialog messages* received from the job submitter and generates dialog messages to be sent to the job submitter. A dialog service comprises at least one *transaction*. In general, a dialog service encompasses at least one dialog step. Exception: in the event of *service chaining*, it is possible for more than one service to comprise a dialog step.

**dialog step**

A dialog step starts when a *dialog message* is received by the *UTM application*. It ends when the UTM application responds.

**dialog terminal process (Unix , Linux and Windows systems)**

A dialog terminal process connects a terminal of a Unix, Linux or Windows system with the work processes of the *UTM application*. Dialog terminal processes are started either when the user enters *utmdtp* or via the LOGIN shell. A separate dialog terminal process is required for each terminal to be connected to a UTM application.

**Distributed Lock Manager / DLM (BS2000 systems)**

Concurrent, cross-computer file accesses can be synchronized using the Distributed Lock Manager.  
DLM is a basic function of HIPLEX® MSCF.

### **distributed processing**

Processing of *dialog jobs* by several different applications or the transfer of *background jobs* to another application. The higher-level protocols *LU6.1* and *OSI TP* are used for distributed processing. openUTM-LU62 also permits distributed processing with LU6.2 partners. A distinction is made between distributed processing with *distributed transactions* (transaction logging across different applications) and distributed processing without distributed transactions (local transaction logging only). Distributed processing is also known as server-server communication.

### **distributed transaction**

*Transaction* which encompasses more than one application and is executed in several different (sub)-transactions in distributed systems.

### **distributed transaction processing**

*Distributed processing with distributed transactions.*

### **dynamic configuration**

Changes to the *configuration* made by the administrator. UTM objects such as *program units*, *transaction codes*, *clients*, *LU6.1 connections*, printers or *user IDs* can be added, modified or in some cases deleted from the configuration while the application is running. To do this, it is necessary to create separate *administration programs* which use the functions of the *program interface for administration*. The WinAdmin administration program or the WebAdmin administration program can be used to do this, or separate *administration programs* must be created that utilize the functions of the *administration program interface*.

### **encryption level**

The encryption level specifies if and to what extent a client message and password are to be encrypted.

### **event-driven service**

This term has been superseded by *event service*.

### **event exit**

Routine in an application program which is started automatically whenever certain events occur (e.g. when a process is started, when a service is terminated). Unlike *event services*, an event exit must not contain any KDCS, CPI-C or XATMI calls.

### **event function**

Collective term for *event exits* and *event services*.

**event service**

*Service* started when certain events occur, e.g. when certain UTM messages are issued. The *program units* for event-driven services must contain KDCS calls.

**filebase**

UTM application filebase

On BS2000 systems, filebase is the prefix for the *KDCFILE*, the *user log file* USLOG and the *system log file* SYSLOG.

On Unix, Linux and Windows systems, filebase is the name of the directory under which the KDCFILE, the user log file USLOG, the system log file SYSLOG and other files relating to the UTM application are stored.

**generation**

See *UTM generation*.

**global secondary storage area**

See *secondary storage area*.

**hardcopy mode**

Operating mode of a printer connected locally to a terminal. Any message which is displayed on screen will also be sent to the printer.

**heterogeneous link**

In the case of *server-server communication*: a link between a *UTM application* and a non-UTM application, e.g. a CICS or TUXEDO application.

**Highly Integrated System Complex / HIPLEX®**

Product family for implementing an operating, load sharing and availability cluster made up of a number of BS2000 servers.

**HIPLEX® MSCF**

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)

Provides the infrastructure and basic functions for distributed applications with HIPLEX®.

**homogeneous link**

In the case of *server-server communication*: a link between two *UTM applications*. It is of no significance whether the applications are running on the same operating system platforms or on different platforms.

**inbound conversation (CPI-C)**

See *incoming conversation*.

### **incoming conversation (CPI-C)**

A conversation in which the local CPI-C program is the *acceptor* is referred to as an incoming conversation. In the X/Open specification, the term “inbound conversation” is used synonymously with “incoming conversation”.

### **initial KDCFILE**

In a *UTM cluster application*, this is the *KDCFILE* generated by *KDCDEF* and which must be copied for each node application before the node applications are started.

### **initiator (CPI-C)**

The communication partners in a *conversation* are referred to as the initiator and the *acceptor*. The initiator sets up the conversation with the CPI-C calls `Initialize_Conversation` and `Allocate`.

### **insert**

Field in a message text in which openUTM enters current values.

### **inverse KDCDEF**

A function which uses the dynamically adapted configuration data in the *KDCFILE* to generate control statements for a *KDCDEF* run. An inverse *KDCDEF* can be started “offline” under *KDCDEF* or “online” via the *program interface for administration*.

### **IUTMDB**

Interface used for the coordinated interaction with resource managers on BS2000 systems. This includes data repositories (LEASY) and data base systems (SESAM/SQL, UDS/SQL).

### **JConnect client**

Designation for clients based on the product openUTM-JConnect. The communication with the UTM application is carried out via the *UPIC protocol*.

### **JDK**

Java Development Kit  
Standard development environment from Oracle Corporation for the development of Java applications.

### **job**

Request for a *service* provided by a *UTM application*. The request is issued by specifying a transaction code. See also: *queued output job*, *dialog job*, *background job*, *job complex*.

**job complex**

Job complexes are used to assign *confirmation jobs* to *asynchronous jobs*. An asynchronous job within a job complex is referred to as a *basic job*.

**job-receiving service (KDCS)**

A job-receiving service is a *service* started by a *job-submitting service* of another server application.

**job-submitting service (KDCS)**

A job-submitting service is a *service* which requests another service from a different server application (*job-receiving service*) in order to process a job.

**KDCADM**

Standard administration program supplied with openUTM. KDCADM provides administration functions which are called with transaction codes (*administration commands*).

**KDCDEF**

UTM tool for the *generation of UTM applications*. KDCDEF uses the configuration information in the KDCDEF control statements to create the UTM objects *KDCFILE* and the ROOT table sources for the main routine *KDCROOT*. In UTM cluster applications, KDCDEF also creates the *cluster configuration file*, the *cluster user file*, the *cluster page pool*, the *cluster GSSB file* and the *cluster ULS file*.

**KDCFILE**

One or more files containing data required for a *UTM application* to run. The KDCFILE is created with the UTM generation tool *KDCDEF*. Among other things, it contains the *configuration* of the application.

**KDCROOT**

Main routine of an *application program* which forms the link between the *program units* and the UTM system code. KDCROOT is linked with the *program units* to form the *application program*.

**KDCS message area**

For KDCS calls: buffer area in which messages or data for openUTM or for the *program unit* are made available.

**KDCS parameter area**

See *parameter area*.

### **KDCS program interface**

Universal UTM program interface compliant with the national DIN 66 265 standard and which includes some extensions. KDCS (compatible data communications interface) allows dialog services to be created, for instance, and permits the use of *message queuing* functions. In addition, KDCS provides calls for *distributed processing*.

### **Kerberos**

Kerberos is a standardized network authentication protocol (RFC1510) based on encryption procedures in which no passwords are sent to the network in clear text.

### **Kerberos principal**

Owner of a key.

Kerberos uses symmetrical encryption, i.e. all the keys are present at two locations, namely with the key owner (principal) and the KDC (Key Distribution Center).

### **key code**

Code that represents specific access authorization or a specific role. Several key codes are grouped into a *key set*.

### **key set**

Group of one or more *key codes* under a particular a name. A key set defines authorization within the framework of the authorization concept used (lock/key code concept or *access list* concept). A key set can be assigned to a *user ID*, an *LTERM partner* an (OSI) *LPAP partner*, a *service* or a *TAC queue*.

### **linkage program**

See *KDCROOT*.

### **local secondary storage area**

See *secondary storage area*.

### **Log4j**

Log4j is part of the Apache Jakarta project. Log4j provides information for logging information (runtime information, trace records, etc.) and configuring the log output. *WS4UTM* uses the software product Log4j for trace and logging functionality.

### **lock code**

Code protecting an LTERM partner or transaction code against unauthorized access. Access is only possible if the *key set* of the accesser contains the appropriate *key code* (lock/key code concept).

**logging process**

Process in Unix, Linux and Windows systems that controls the logging of account records or monitoring data.

**LPAP bundle**

LPAP bundles allow messages to be distributed to LPAP partners across several partner applications. If a UTM application has to exchange a very large number of messages with a partner application then load distribution may be improved by starting multiple instances of the partner application and distributing the messages across the individual instances. In an LPAP bundle, *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 UTM generation. LPAP bundles exist for both the OSI TP protocol and the LU6.1 protocol.

**LPAP partner**

In the case of *distributed processing* via the *LU6.1* protocol, an LPAP partner for each partner application must be configured in the local application. The LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned LPAP partner and not by the application name or address.

**LTERM bundle**

An LTERM bundle (connection bundle) consists of a master LTERM and multiple slave LTERMs. An LTERM bundle (connection bundle) allows you to distribute queued messages to a logical partner application evenly across multiple parallel connections.

**LTERM group**

An LTERM group consists of one or more alias LTERMs, the group LTERMs and a primary LTERM. In an LTERM group, you assign multiple LTERMs to a connection.

**LTERM partner**

LTERM partners must be configured in the application if you want to connect clients or printers to a *UTM application*. A client or printer can only be connected if an LTERM partner with the appropriate properties is assigned to it. This assignment is generally made in the *configuration*, but can also be made dynamically using terminal pools.

### **LTERM pool**

The TPOOL statement allows you to define a pool of LTERM partners instead of issuing one LTERM and one PTERM statement for each *client*. If a client establishes a connection via an LTERM pool, an LTERM partner is assigned to it dynamically from the pool.

### **LU6.1**

Device-independent data exchange protocol (industrial standard) for transaction-oriented *server-server communication*.

### **LU6.1-LPAP bundle**

*LPAP bundle* for *LU6.1* partner applications.

### **LU6.1 partner**

Partner of the *UTM application* that communicates with the UTM application via the *LU6.1* protocol.

Examples of this type of partner are:

- a UTM application that communicates via *LU6.1*
- an application in the IBM environment (e.g. CICS, IMS or TXSeries) that communicates via *LU6.1*

### **main process (Unix systems / Windows systems)**

Process which starts the *UTM application*. It starts the *work processes*, the *UTM system processes*, *printer processes*, *network processes*, *logging process* and the *timer process* and monitors the *UTM application*.

### **main routine KDCROOT**

See *KDCROOT*.

### **management unit**

*SE Servers component*; in combination with the *SE Manager*, permits centralized, web-based management of all the units of an SE server.

### **message definition file**

The message definition file is supplied with openUTM and, by default, contains the UTM message texts in German and English together with the definitions of the message properties. Users can take this file as a basis for their own message modules.



**message destination**

Output medium for a *message*. Possible message destinations for a message from the openUTM transaction monitor include, for instance, terminals, *TS applications*, the *event service* MSGTAC, the *system log file* SYSLOG or *TAC queues*, *asynchronous TACs*, *USER queues*, SYSOUT/SYSLST or stderr/stdout. The message destinations for the messages of the UTM tools are SYSOUT/SYSLST and stderr/stdout.

**message queue**

Queue in which specific messages are kept with transaction management until further processed. A distinction is drawn between *service-controlled queues* and *UTM-controlled queues*, depending on who monitors further processing.

**message queuing**

Message queuing (MQ) is a form of communication in which the messages are exchanged via intermediate queues rather than directly. The sender and recipient can be separated in space or time. The transfer of the message is independent of whether a network connection is available at the time or not. In openUTM there are *UTM-controlled queues* and *service-controlled queues*.

**MSGTAC**

Special event service that processes messages with the message destination MSGTAC by means of a program. MSGTAC is an asynchronous service and is created by the operator of the application.

**multiplex connection (BS2000 systems)**

Special method offered by *OMNIS* to connect terminals to a *UTM application*. A multiplex connection enables several terminals to share a single transport connection.

**multi-step service (KDCS)**

*Service* carried out in a number of *dialog steps*.

**multi-step transaction**

*Transaction* which comprises more than one *processing step*.

**Network File System/Service / NFS**

Allows Unix systems to access file systems across the network.

**network process (Unix systems / Windows systems)**

A process in a *UTM application* for connection to the network.

### **network selector**

The network selector identifies a service access point to the network layer of the *OSI reference model* in the local system.

### **node**

Individual computer of a *cluster*.

### **node application**

*UTM application* that is executed on an individual *node* as part of a *UTM cluster application*.

### **node bound service**

A node bound service belonging to a user can only be continued at the node application at which the user was last signed on. The following services are always node bound:

- Services that have started communications with a job receiver via LU6.1 or OSI TP and for which the job-receiving service has not yet been terminated
- Inserted services in a service stack
- Services that have completed a SESAM transaction

In addition, a user's service is node bound as long as the user is signed-on at a node application.

### **node filebase**

Filename prefix or directory name for the *node application's KDCFILE, user log file* and *system log file*.

### **node recovery**

If a node application terminates abnormally and no rapid warm start of the application is possible on its associated *node computer* then it is possible to perform a node recovery for this node on another node in the UTM cluster. In this way, it is possible to release locks resulting from the failed node application in order to prevent unnecessary impairments to the running *UTM cluster application*.

### **normal termination of a UTM application**

Controlled termination of a *UTM application*. Among other things, this means that the administration data in the *KDCFILE* are updated. The *administrator* initiates normal termination (e.g. with *KDCSHUT N*). After a normal termination, *openUTM* carries out any subsequent start as a *cold start*.

### **object identifier**

An object identifier is an identifier for objects in an OSI environment which is unique throughout the world. An object identifier comprises a sequence of integers which represent a path in a tree structure.

**OMNIS (BS2000 systems)**

OMNIS is a “session manager” which lets you set up connections from one terminal to a number of partners in a network concurrently. OMNIS also allows you to work with multiplex connections.

**online import**

In a *UTM cluster application*, online import refers to the import of application data from a normally terminated node application into a running node application.

**online update**

In a *UTM cluster application*, online update refers to a change to the application configuration or the application program or the use of a new UTM revision level while a *UTM cluster application* is running.

**open terminal pool**

*Terminal pool* which is not restricted to clients of a single computer or particular type. Any client for which no computer- or type-specific terminal pool has been generated can connect to this terminal pool.

**OpenCPIC**

Carrier system for UTM clients that use the *OSI TP* protocol.

**OpenCPIC client**

*OSI TP* partner application with the *OpenCPIC* carrier system.

**openSM2**

The openSM2 product line offers a consistent solution for the enterprise-wide performance management of server and storage systems. openSM2 offers the acquisition of monitoring data, online monitoring and offline evaluation.

**openUTM 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 *OpenCPIC client*
- applications from other TP monitors that support OSI TP

### **outbound conversation (CPI-C)**

See *outgoing conversation*.

### **outgoing conversation (CPI-C)**

A conversation in which the local CPI-C program is the *initiator* is referred to as an outgoing conversation. In the X/Open specification, the term “outbound conversation” is used synonymously with “outgoing conversation”.

**page pool**

Part of the *KDCFILE* in which user data is stored.

In a *standalone application* this data consists, for example, of *dialog messages*, messages sent to *message queues*, *secondary memory areas*.

In a UTM cluster application, it consists, for example, of messages to *message queues*, *TLS*.

**parameter area**

Data structure in which a program unit passes the operands required for a UTM call to *openUTM*.

**partner application**

Partner of a UTM application during *distributed processing*. Higher communication protocols are used for distributed processing (*LU6.1*, *OSI TP* or *LU6.2* via the *openUTM-LU62* gateway).

**postselection (BS2000 systems)**

Selection of logged UTM events from the SAT logging file which are to be evaluated. Selection is carried out using the *SATUT* tool.

**prepare to commit (PTC)**

Specific state of a distributed transaction

Although the end of the distributed transaction has been initiated, the system waits for the partner to confirm the end of the transaction.

**preselection (BS2000 systems)**

Definition of the UTM events which are to be logged for the *SAT audit*.

Preselection is carried out with the UTM-SAT administration functions. A distinction is made between event-specific, user-specific and job-specific (TAC-specific) preselection.

**presentation selector**

The presentation selector identifies a service access point to the presentation layer of the *OSI reference model* in the local system.

**primary storage area**

Area in main memory to which the *KDCS program unit* has direct access, e.g. *standard primary working area*, *communication area*.

**print administration**

Functions for *print control* and the administration of *queued output jobs*, sent to a printer.

**print control**

openUTM functions for controlling print output.

**printer control LTERM**

A printer control LTERM allows a client or terminal user to connect to a UTM application. The printers assigned to the printer control LTERM can then be administered from the client program or the terminal. No administration rights are required for these functions.

**printer control terminal**

This term has been superseded by *printer control LTERM*.

**printer group (Unix systems)**

For each printer, a Unix system sets up one printer group by default that contains this one printer only. It is also possible to assign several printers to one printer group or to assign one printer to several different printer groups.

**printer pool**

Several printers assigned to the same *LTERM partner*.

**printer process (Unix systems)**

Process set up by the *main process* for outputting *asynchronous messages* to a *printer group*. The process exists as long as the printer group is connected to the *UTM application*. One printer process exists for each connected printer group.

**process**

The openUTM manuals use the term “process” as a collective term for processes (Unix systems / Windows systems) and tasks (BS2000 systems).

**processing step**

A processing step starts with the receipt of a *dialog message* sent to the *UTM application* by a *client* or another server application. The processing step ends either when a response is sent, thus also terminating the *dialog step*, or when a dialog message is sent to a third party.

**program interface for administration**

UTM program interface which helps users to create their own *administration programs*. Among other things, the program interface for administration provides functions for *dynamic configuration*, for modifying properties and application parameters and for querying information on the configuration and the current workload of the application.

**program space (BS2000 systems)**

Virtual address space of BS2000 which is divided into memory classes and in which both executable programs and pure data are addressed.

**program unit**

UTM *services* are implemented in the form of one or more program units. The program units are components of the *application program*. Depending on the employed API, they may have to contain KDCS, XATMI or CPIC calls. They can be addressed using *transaction codes*. Several different transaction codes can be assigned to a single program unit.

**queue**

See *message queue*.

**queued output job**

Queued output jobs are *asynchronous jobs* which output a message, such as a document, to a printer, a terminal or a transport system application. Queued output jobs are processed by UTM system functions exclusively, i.e. it is not necessary to create program units to process them.

**Quick Start Kit**

A sample application supplied with openUTM (Windows systems).

**redelivery**

Repeated delivery of an *asynchronous message* that could not be processed correctly because, for example, the *transaction* was rolled back or the *asynchronous service* was terminated abnormally. The message is returned to the message queue and can then be read and/or processed again.

**reentrant program**

Program whose code is not altered when it runs. On BS2000 systems this constitutes a prerequisite for using *shared code*.

**request**

Request from a *client* or another server for a *service function*.

**requestor**

In XATMI, the term requestor refers to an application which calls a service.

### resource manager

Resource managers (RMs) manage data resources. Database systems are examples of resource managers. openUTM, however, also provides its own resource managers for accessing message queues, local memory areas and logging files, for instance. Applications access RMs via special resource manager interfaces. In the case of database systems, this will generally be SQL and in the case of openUTM RMs, it is the KDCS interface.

### restart

See *screen restart*,  
see *service restart*.

### RFC1006

A protocol defined by the IETF (Internet Engineering Task Force) belonging to the TCP/IP family that implements the ISO transport services (transport class 0) based on TCP/IP.

### RSA

Abbreviation for the inventors of the RSA encryption method (Rivest, Shamir and Adleman). This method uses a pair of keys that consists of a public key and a private key. A message is encrypted using the public key, and this message can only be decrypted using the private key. The pair of RSA keys is created by the UTM application.

### SAT audit (BS2000 systems)

*Audit* carried out by the SAT (Security Audit Trail) component of the BS2000 software product SECOS.

### screen restart

If a *dialog service* is interrupted, openUTM again displays the *dialog message* of the last completed *transaction* on screen when the service restarts provided that the last transaction output a message on the screen.

### SE manager

Web-based graphical user interface (GUI) for the SE series of Business Servers. SE Manager runs on the *management unit* and permits the central operation and administration of server units (with /390 architecture and/or x86 architecture), application units (x86 architecture), net unit and peripherals.

### SE server

A Business Server from Fujitsu's SE series.



**secondary storage area**

Memory area secured by transaction logging and which can be accessed by the KDCS *program unit* with special calls. Local secondary storage areas (LSSBs) are assigned to one *service*. Global secondary storage areas (GSSBs) can be accessed by all services in a *UTM application*. Other secondary storage areas include the *terminal-specific long-term storage (TLS)* and the *user-specific long-term storage (ULS)*.

**selector**

A selector identifies a service access point to services of one of the layers of the *OSI reference model* in the local system. Each selector is part of the address of the access point.

**semaphore (Unix systems / Windows systems)**

Unix systems and Windows systems resource used to control and synchronize processes.

**server**

A server is an *application* which provides *services*. The computer on which the applications are running is often also referred to as the server.

**server-server communication**

See *distributed processing*.

**server side of a conversation (CPI-C)**

This term has been superseded by *acceptor*.

**service**

Services process the *jobs* that are sent to a server application. A service of a UTM application comprises one or more transactions. The service is called with the *service TAC*. Services can be requested by *clients* or by other servers.

**service access point**

In the OSI reference model, a layer has access to the services of the layer below at the service access point. In the local system, the service access point is identified by a *selector*. During communication, the *UTM application* links up to a service access point. A connection is established between two service access points.

**service chaining (KDCS)**

When service chaining is used, a follow-up service is started without a *dialog message* specification after a *dialog service* has completed .

### **service-controlled queue**

*Message queue* in which the calling and further processing of messages is controlled by *services*. A service must explicitly issue a KDCS call (DGET) to read the message. There are service-controlled queues in openUTM in the variants *USER queue*, *TAC queue* and *temporary queue*.

### **service restart (KDCS)**

If a service is interrupted, e.g. as a result of a terminal user signing off or a *UTM application* being terminated, openUTM carries out a *service restart*. An *asynchronous service* is restarted or execution is continued at the most recent *synchronization point*, and a *dialog service* continues execution at the most recent *synchronization point*. As far as the terminal user is concerned, the service restart for a dialog service appears as a *screen restart* provided that a dialog message was sent to the terminal user at the last synchronization point.

### **service routine**

See *program unit*.

### **service stacking (KDCS)**

A terminal user can interrupt a running *dialog service* and insert a new dialog service. When the inserted *service* has completed, the interrupted service continues.

### **service TAC (KDCS)**

Transaction code used to start a *service*.

### **session**

Communication relationship between two addressable units in the network via the SNA protocol *LU6.1*.

### **session selector**

The session selector identifies an *access point* in the local system to the services of the session layer of the *OSI reference model*.

### **shared code (BS2000 systems)**

Code which can be shared by several different processes.

### **shared memory**

Virtual memory area which can be accessed by several different processes simultaneously.

**shared objects (Unix systems / Windows systems)**

Parts of the *application program* can be created as shared objects. These objects are linked to the application dynamically and can be replaced during live operation. Shared objects are defined with the KDCDEF statement SHARED-OBJECT.

**sign-on check**

See *system access control*.

**sign-on service (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)

SOA is a system architecture concept in which functions are implemented in the form of re-usable, technically independent, loosely coupled *services*. Services can be called independently of the underlying implementations via interfaces which may possess public and, consequently, trusted specifications. Service interaction is performed via a communication infrastructure made available for this purpose.

**SOAP**

SOAP (Simple Object Access Protocol) is a protocol used to exchange data between systems and run remote procedure calls. SOAP also makes use of the services provided by other standards, XML for the representation of the data and Internet transport and application layer protocols for message transfer.

**socket connection**

Transport system connection that uses the socket interface. The socket interface is a standard program interface for communication via TCP/IP.

**standalone application**

See *standalone UTM application*.

**standalone UTM application**

Traditional *UTM application* that is not part of a *UTM cluster application*.

**standard primary working area (KDCS)**

Area in main memory available to all KDCS *program units*. The contents of the area are either undefined or occupied with a fill character when the program unit starts execution.

**start format**

Format output to a terminal by openUTM when a user has successfully signed on to a *UTM application* (except after a *service restart* and during sign-on via the *sign-on service*).

**static configuration**

Definition of the *configuration* during generation using the UTM tool *KDCDEF*.

**SYSLOG file**

See *system log file*.

**synchronization point, consistency point**

The end of a *transaction*. At this time, all the changes made to the *application information* during the transaction are saved to prevent loss in the event of a crash and are made visible to others. Any locks set during the transaction are released.

**system access control**

A check carried out by openUTM to determine whether a certain *user ID* is authorized to work with the *UTM application*. The authorization check is not carried out if the UTM application was generated without user IDs.

**system log file**

File or file generation to which openUTM logs all UTM messages for which SYSLOG has been defined as the *message destination* during execution of a *UTM application*.

**TAC**

See *transaction code*.

**TAC queue**

*Message queue* generated explicitly by means of a KDCDEF statement. A TAC queue is a *service-controlled queue* that can be addressed from any service using the generated name.

**temporary queue**

*Message queue* created dynamically by means of a program that can be deleted again by means of a program (see *service-controlled queue*).

**terminal-specific long-term storage (KDCS)**

*Secondary storage area* assigned to an *LTERM*, *LPAP* or *OSI-PAP partner* and which is retained after the application has terminated.

**time-driven job**

*Job* which is buffered by openUTM in a *message queue* up to a specific time until it is sent to the recipient. The recipient can be an *asynchronous service* of the same application, a *TAC queue*, a partner application, a terminal or a printer. Time-driven jobs can only be issued by KDCS *program units*.

**timer process (Unix systems / Windows systems)**

Process which accepts jobs for controlling the time at which *work processes* are executed. It does this by entering them in a job list and releasing them for processing after a time period defined in the job list has elapsed.

**TNS (Unix systems / Windows systems)**

Abbreviation for the Transport Name Service. TNS assigns a transport selector and a transport system to an application name. The application can be reached through the transport system.

**Tomcat**

see *Apache Tomcat*

**transaction**

Processing section within a *service* for which adherence to the *ACID properties* is guaranteed. If, during the course of a transaction, changes are made to the *application information*, they are either made consistently and in their entirety or not at all (all-or-nothing rule). The end of the transaction forms a *synchronization point*.

**transaction code/TAC**

Name which can be used to identify a *program unit*. The transaction code is assigned to the program unit during *static* or *dynamic configuration*. It is also possible to assign more than one transaction code to a program unit.

**transaction rate**

Number of *transactions* successfully executed per unit of time.

**transfer syntax**

With *OSI TP*, the data to be transferred between two computer systems is converted from the local format into transfer syntax. Transfer syntax describes the data in a neutral format which can be interpreted by all the partners involved. An *Object Identifier* must be assigned to each transfer syntax.

### **transport selector**

The transport selector identifies a service access point to the transport layer of the *OSI reference model* in the local system.

### **transport system application**

Application which is based directly on a transport system interface (e.g. CMX, DCAM or socket). When transport system applications are connected, the partner type APPLI or SOCKET must be specified during *configuration*. A transport system application cannot be integrated in a *distributed transaction*.

### **TS application**

See *transport system application*.

### **typed buffer (XATMI)**

Buffer for exchanging typed and structured data between communication partners. Typed buffers ensure that the structure of the exchanged data is known to both partners implicitly.

### **UPIC**

Carrier system for openUTM clients. UPIC stands for Universal Programming Interface for Communication. The communication with the UTM application is carried out via the *UPIC protocol*.

### **UPIC Analyzer**

Component used to analyze the UPIC communication recorded with *UPIC Capture*. This step is used to prepare the recording for playback using *UPIC Replay*.

### **UPIC Capture**

Used to record communication between UPIC clients and UTM applications so that this can be replayed subsequently (*UPIC Replay*).

### **UPIC client**

The designation for openUTM clients with the UPIC carrier system and for *JConnect clients*.

### **UPIC protocol**

Protocol for the client server communication with *UTM applications*. The UPIC protocol is used by *UPIC clients* and *JConnect clients*.

### **UPIC Replay**

Component used to replay the UPIC communication recorded with *UPIC Capture* and prepared with *UPIC Analyzer*.

**user exit**

This term has been superseded by *event exit*.

**user ID**

Identifier for a user defined in the *configuration* for the *UTM application* (with an optional password for *system access control*) and to whom special data access rights (*system access control*) have been assigned. A terminal user must specify this ID (and any password which has been assigned) when signing on to the UTM application. On BS2000 systems, system access control is also possible via *Kerberos*.

For other clients, the specification of a user ID is optional, see also *connection user ID*.

UTM applications can also be generated without user IDs.

**user log file**

File or file generation to which users write variable-length records with the KDCS LPUT call. The data from the KB header of the *KDCS communication area* is prefixed to every record. The user log file is subject to transaction management by openUTM.

**USER queue**

*Message queue* made available to every user ID by openUTM. A USER queue is a *service-controlled queue* and is always assigned to the relevant user ID. You can restrict the access of other UTM users to your own USER queue.

**user-specific long-term storage**

*Secondary storage area* assigned to a *user ID*, a *session* or an *association* and which is retained after the application has terminated.

**USLOG file**

See *user log file*.

**UTM application**

A UTM application provides *services* which process jobs from *clients* or other applications. openUTM is responsible for transaction logging and for managing the communication and system resources. From a technical point of view, a UTM application is a process group which forms a logical server unit at runtime.

**UTM client**

See *client*.

### 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 (APPLIMODE=FAST), application data that is valid throughout the cluster is also saved. In this case, GSSB and ULS data is treated in exactly the same way as in UTM cluster applications generated with UTM-S. However, service data relating to users with RESTART=YES is written only when the relevant user signs off and not at the end of each transaction.

### UTM generation

*Static configuration* of a *UTM application* using the UTM tool KDCDEF and creation of an application program.



**UTM message**

Messages are issued to *UTM message destinations* by the openUTM transaction monitor or by UTM tools (such as *KDCDEF*). A message comprises a message number and a message text, which can contain *inserts* with current values. Depending on the message destination, either the entire message is output or only certain parts of the message, such as the inserts).

**UTM page**

A UTM page is a unit of storage with a size of either 2K, 4K or 8 K. In *standalone UTM applications*, the size of a UTM page on generation of the UTM application can be set to 2K, 4K or 8 K. The size of a UTM page in a *UTM cluster application* is always 4K or 8 K. The *page pool* and the restart area for the *KDCFILE* and *UTM cluster files* are divided into units of the size of a UTM page.

**utmpath (Unix systems / Windows systems)**

The directory under which the openUTM components are installed is referred to as *utmpath* in this manual.

To ensure that openUTM runs correctly, the environment variable *UTMPATH* must be set to the value of *utmpath*. On Unix and Linux systems, you must set *UTMPATH* before a UTM application is started. On Windows systems *UTMPATH* is set in accordance with the UTM version installed most recently.

**UTM-S**

In the case of UTM-S applications, openUTM saves all user data as well as the administration data beyond the end of an application and any system crash which may occur. In addition, UTM-S guarantees the security and consistency of the application data in the event of any malfunction. UTM applications are usually generated as UTM-S applications (UTM secure).

**UTM SAT administration (BS2000 systems)**

UTM-SAT administration functions control which UTM events relevant to security which occur during operation of a *UTM application* are to be logged by *SAT*. Special authorization is required for UTM-SAT administration.

**UTM system process**

UTM process that is started in addition to the processes specified via the start parameters and which only handles selected jobs. UTM system processes ensure that UTM applications continue to be reactive even under very high loads.

**UTM terminal**

This term has been superseded by *LTERM partner*.

### **UTM tool**

Program which is provided together with openUTM and which is needed for UTM specific tasks (e.g for configuring).

### **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.

### **WebAdmin**

Web-based tool for the administration of openUTM applications via a Web browser. WebAdmin includes not only the full function scope of the *administration program interface* but also additional functions.

### **Web service**

Application which runs on a Web server and is (publicly) available via a standardized, programmable interface. Web services technology makes it possible to make UTM program units available for modern Web client applications independently of the programming language in which they were developed.

### **WinAdmin**

Java-based tool for the administration of openUTM applications via a graphical user interface. WinAdmin includes not only the full function scope of the *administration program interface* but also additional functions.

### **work process (Unix systems / Windows systems)**

A process within which the *services* of a *UTM application* run.

**workload capture & replay**

Family of programs used to simulate load situations; consisting of the main components *UPIC Capture*, *UPIC Analyzer* and *Upic Replay* and - on Unix, Linux and Windows systems - the utility program *kdcsort*. Workload Capture & Replay can be used to record UPIC sessions with UTM applications, analyze these and then play them back with modified load parameters.

**WS4UTM**

WS4UTM (**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 systems)**

XHCS (Extended Host Code Support) is a BS2000 software product providing support for international character sets.

**XML**

XML (eXtensible Markup Language) is a metalanguage standardized by the W3C (WWW Consortium) in which the interchange formats for data and the associated information can be defined.



---

# Abbreviations

Please note: Some of the abbreviations used here derive from the German acronyms used in the original German product(s).

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AES	Advanced Encryption Standard
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder - Loader - Starter (BS2000 systems)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Coded Character Set
CCSN	Coded Character Set Name
CICS	Customer Information Control System
CID	Control Identification
CMX	Communication Manager in Unix, Linux and Windows Systems
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000 systems)
DB	Database
DC	Data Communication
DCAM	Data Communication Access Method

DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000 systems)
DMS	Data Management System
DNS	Domain Name Service
DP	Distributed Processing
DSS	Terminal (Datensichtstation)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans <sup>TM</sup>
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GSSB	Global Secondary Storage Area
HIPLEX <sup>®</sup>	Highly Integrated System Complex (BS2000 systems)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IFG	Interactive Format Generator
ILCS	Inter-Language Communication Services (BS2000 systems)
IMS	Information Management System (IBM)
IPC	Inter-Process Communication
IRV	International Reference Version
ISO	International Organization for Standardization
Java EE	Java Platform, Enterprise Edition
JCA	Java EE Connector Architecture
JDK	Java Development Kit
KAA	KDCS Application Area
KB	Communication Area
KBPRG	KB Program Area
KCADMI	KDC Administration Interface
KDCS	Compatible Data Communication Interface

KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000 systems)
LSSB	Local Secondary Storage Area
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000 systems)
NB	Message Area
NEA	Network Architecture for BS2000 Systems
NFS	Network File System/Service
NLS	Native Language Support
OLTP	Online Transaction Processing
OML	Object Module Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Process Identification
PIN	Personal Identification Number
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Computer Center Accounting Procedure
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption algorithm according to Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000 systems)
RTS	Runtime System
SAT	Security Audit Trail (BS2000 systems)
SECOS	Security Control System
SEM	SE Manager
SGML	Standard Generalized Markup Language
SLU	Secondary Logical Unit

## Abbreviations

---

SM2	Software Monitor 2
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primary Working Area
SQL	Structured Query Language
SSB	Secondary Storage Area
SSO	Single Sign-On
TAC	Transaction Code
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-Specific Long-Term Storage
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaction Mode)
TPR	Privileged Function State in BS2000 systems (Task Privileged)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Non-Privileged Function State in BS2000 systems (Task User)
TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universal Database System
UDT	Unstructured Data Transfer
ULS	User-Specific Long-Term Storage
UPIC	Universal Programming Interface for Communication
USP	UTM Socket Protocol
UTM	Universal Transaction Monitor
UTM-D	UTM Variant for Distributed Processing in BS2000 systems
UTM-F	UTM Fast Variant
UTM-S	UTM Secure Variant
UTM-XML	openUTM XML Interface



VGID	Service ID
VTSU	Virtual Terminal Support
WAN	Wide Area Network
WS4UTM	Web-Services for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (X/Open interface for 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

You will find the manuals on the internet at <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**  
**Generating Applications**  
User Guide

**openUTM**  
**Using UTM Applications on BS2000 Systems**  
User Guide

**openUTM**  
**Using UTM Applications on Unix, Linux and Windows Systems**  
User Guide

**openUTM**  
**Administering Applications**  
User Guide

**openUTM**  
**Messages, Debugging and Diagnostics on BS2000 Systems**  
User Guide

**openUTM**  
**Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems**  
User Guide

**openUTM**  
**Creating Applications with X/Open Interfaces**  
User Guide

**openUTM**  
**XML for openUTM**

**openUTM Client (Unix systems)**  
**for the OpenCPIC Carrier System**  
**Client-Server Communication with openUTM**  
User Guide

**openUTM Client**  
**for the UPIC Carrier System**  
**Client-Server Communication with openUTM**  
User Guide

**openUTM WinAdmin**  
**Graphical Administration Workstation for openUTM**  
Description and online help system

**openUTM WebAdmin**  
**Web Interface for Administering openUTM**  
Description and online help system

**openUTM, openUTM-LU62**  
**Distributed Transaction Processing**  
**between openUTM and CICS, IMS and LU6.2 Applications**  
User Guide

**openUTM (BS2000)**  
**Programming Applications with KDCS for Assembler**  
Supplement to Core Manual

**openUTM (BS2000)**  
**Programming Applications with KDCS for Fortran**  
Supplement to Core Manual

**openUTM** (BS2000)  
**Programming Applications with KDCS for Pascal-XT**  
Supplement to Core Manual

**openUTM** (BS2000)  
**Programming Applications with KDCS for PL/I**  
Supplement to Core Manual

**WS4UTM** (Unix systems and Windows systems)  
**WebServices for openUTM**

## Documentation for the openSEAS product environment

**BeanConnect**  
User Guide

**openUTM-JConnect**  
**Connecting Java Clients to openUTM**  
User documentation and Java docs

**WebTransactions**  
**Concepts and Functions**

**WebTransactions**  
**Template Language**

**WebTransactions**  
**Web Access to openUTM Applications via UPIC**

**WebTransactions**  
**Web Access to MVS Applications**

**WebTransactions**  
**Web Access to OSD Applications**

## Documentation for the BS2000 environment

**AID**  
**Advanced Interactive Debugger**  
**Core Manual**  
User Guide

**AID**  
**Advanced Interactive Debugger**  
**Debugging of COBOL Programs**  
User Guide

**AID**  
**Advanced Interactive Debugger**  
**Debugging of C/C++ Programs**  
User Guide

**BCAM**  
**BCAM Volume 1/2**  
User Guide

**BINDER**  
User Guide

**BS2000 OSD/BC**  
**Commands Volume 1 - 7**  
User Guide

**BS2000 OSD/BC**  
**Executive Macros**  
User Guide

**BS2IDE**  
Eclipse-based Integrated Development Environment for BS2000  
User Guide and Installation Guide  
Web page: <https://bs2000.ts.fujitsu.com/bs2ide/>

**BLSSERV**  
**Dynamic Binder Loader / Starter in BS2000/OSD**  
User Guide

**DCAM**  
**COBOL Calls**  
User Guide

**DCAM**  
**Macros**  
User Guide

**DCAM**  
**Program Interfaces**  
Description

**FHS**  
**Format Handling System for openUTM, TIAM, DCAM**  
User Guide

**IFG for FHS**  
User Guide

**HIPLEX AF**  
**High-Availability of Applications in BS2000/OSD**  
Product Manual

**HIPLEX MSCF**  
**BS2000 Processor Networks**  
User Guide

**IMON**  
**Installation Monitor**  
User Guide

**MT9750 (MS Windows)**  
**9750 Emulation under Windows**  
Product Manual

**OMNIS/OMNIS-MENU**  
**Functions and Commands**  
User Guide

**OMNIS/OMNIS-MENU**  
**Administration and Programming**  
User Guide

**OSS (BS2000)**  
**OSI Session Service**  
User Guide

**openSM2**  
**Software Monitor**  
User Guide

**RSO**  
**Remote SPOOL Output**  
User Guide

**SECOS**  
**Security Control System**  
User Guide

**SECOS**  
**Security Control System**  
Ready Reference

**SESAM/SQL**  
**Database Operation**  
User Guide

**TIAM**  
User Guide

**UDS/SQL**  
**Database Operation**  
User Guide

**Unicode in BS2000/OSD**  
Introduction

**VTSU**  
**Virtual Terminal Support**  
User Guide

**XHCS**  
**8-Bit Code and Unicode Support in BS2000/OSD**  
User Guide



## Documentation for the Unix, Linux and Windows system environment

**CMX V6.0** (Unix systems)

**Betrieb und Administration** (only available in German)

User Guide

**CMX V6.0**

Programming CMX Applications

Programming Guide

**OSS (UNIX)**

**OSI Session Service**

User Guide

PRIMECLUSTER<sup>TM</sup>

**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 CAE Specification  
Distributed Transaction Processing:  
The CPI-C Specification, Version 2  
ISBN 1 85912 135 7

### **Reference Model**

X/Open Guide  
Distributed Transaction Processing:  
Reference Model, Version 2  
ISBN 1 85912 019 9

### **TX**

X/Open CAE Specification  
Distributed Transaction Processing:  
The TX (Transaction Demarcation) Specification  
ISBN 1 85912 094 6

### **XATMI**

X/Open CAE Specification  
Distributed Transaction Processing  
The XATMI Specification  
ISBN 1 85912 130 6

### **XML**

W3C specification (www consortium)  
Web page: <http://www.w3.org/XML>

---

# Index

## A

acceptor 38  
address information (conversation) 47  
administration journal 164  
alignment 110  
ASCII conversion 113  
ASN.1 type 112  
association 38, 42  
    characteristics 47  
    generation specifications 98  
    maximum number 55  
associations (XATMI) 131  
asynchronous conversation 39, 83  
asynchronous request-response paradigm 107  
asynchronous single request 108  
asynchronous TAC 109  
automatic conversion 113

## B

big endian 110  
buffer  
    define 126  
buffer size  
    maximum 121  
    maximum, CPI-C 57  
BUFFER statement 126

## C

C data types 111  
CAE (Common Application Environment) 27  
call  
    service 117  
carrier system  
    automatic conversion 113  
character set 57

character set conversion 113  
character set encoding 113  
characteristic  
    commit\_return 150  
    conversation\_type 46  
    conversion of 57  
    transaction\_control 149  
    XATMI 121  
client, XATMI 103  
client side of conversation see initiator  
client/server link  
    CPI-C, XATMI 33  
cluster administration journal 164  
code  
    data types 112, 127  
code conversion tables 60  
comment line  
    LCF 122  
commit\_return characteristic 150  
Common Application Environment (CAE) 27  
common data types 111  
communication paradigm  
    asynchronous request-response 107  
    conversational 109  
    single request 108, 109  
    synchronous request-response 107  
communication partner, CPI-C 40  
communication resource manager 103  
Communication Resource Manager (CRM) 28  
COMP 126  
compiler 126  
configure XATMI 122  
Confirm-Deallocate (conversation state) 64  
Confirm-Send (conversation state) 64  
confirmation 56

- confirmation request (CIPIC) 56
  - continuation character 122
  - conversation 38, 109
    - acceptor 38
    - asynchronous 39
    - dialog 39
    - half-duplex 53
    - initiator 38
    - maximum number 121
    - multiple in one program 54
  - conversation characteristic 39
    - addressing 52
    - conversation\_type 46
    - sync\_level 56
  - conversation ID 39
  - conversation states 64
  - conversation type 46
  - conversational 124, 126
  - conversational paradigm 109
  - conversion 113
  - conversion tables 60
  - convert
    - characteristics 57
  - convert user data 57
  - COPY element
    - CPI-C 95
    - directory 35
    - TX 156
    - XATMI 137
  - CPI-C
    - address communication partners 47
    - client/server link 33
    - communication partners 40
    - flowchart 44
    - Integration in openUTM 31
    - linking applications 42
    - sample application 44
    - server/server link 32
    - standard 37
  - CPI-C application
    - create 95
    - generate 97
    - link 95, 96
  - CPI-C calls
    - available in UTM 66
    - not available in UTM 82
    - UTM-specific features 70
  - CPI-C program
    - error diagnosis 100
    - start 95
    - with asynchronous transaction code 39, 83
    - with dialog transaction code 39, 83
  - CPI-C program unit 95
  - CPI-C program with asynchronous transaction code 83
  - create
    - LCF 128
    - TX application 156
    - XATMI application 137
  - create CPI-C application 95
  - CRM (Communication Resource Manager) 28
- D**
- data buffer, XATMI 111
  - data structure
    - name 126
  - data transfer
    - service routine 117
  - data types, XATMI 111
  - data volumes, large 121
  - define service 122
  - dialog conversation 39, 83
  - Distributed Transaction Processing (DTP) 28
  - documentation
    - summary 10
  - DTP (Distributed Transaction Processing) 28
  - DTP reference model 28
- E**
- EBCDIC conversion
    - CPI-C 57
    - XATMI 113
  - end service 103, 124
  - error diagnosis
    - TX 157
    - XATMI 139
  - error handling 118

**F**

fields of application (interfaces) 34  
flowchart, CPI-C 44

**G**

generate  
  CPI-C application 97  
  LCF 128  
  XATMI application 131  
global transaction  
  CPIC and TX 151  
  XATMI and TX 153

**H**

half-duplex conversation 53  
Heterogeneous link 59  
Homogenous link 58  
HP-UX 9

**I**

inbound conversation 38  
include file  
  Directory 35  
  for CPI-C 96  
  for CPIC 95  
  for TX 156  
  for XATMI 137  
incoming conversation 38, 54  
Initialize (conversation state) 64  
initiator 38  
interfaces in openUTM 30  
intermediate service 103  
internal-service-name 123, 125

**J**

job variable  
  XATMI 139, 140

**K**

KDCDEF generation  
  XATMI 131  
KDCS program interface 29

**L**

large data volumes 121

LC definition file 128  
LC description files 128  
LCF 105, 122, 128  
  create 128  
  generate 128  
library for X/Open interfaces 35  
linking  
  CPI-C application 95, 96  
linking client/server  
  XATMI 104  
Linking client/server with CPI-C and XTAMI 33  
linking server/server 42  
Linux distribution 9  
little endian 110  
local configuration  
  code for syntax 112  
local configuration definition file 122  
local configuration file 105, 128  
  create 128  
  generate 128  
LTAC  
  XATMI 99, 133  
LTAC statement  
  CPIC 47  
LU6.1 protocol 28, 31  
LU6.1 session 42

**M**

machine dependencies 110  
mapped conversation 46  
MAX NB, XATMI 121  
maximum buffer size 121  
maximum lengths 121  
maximum message length 121  
message length  
  maximum, CPI-C 57  
  maximum, XATMI 121  
messages  
  XATMI 144  
  xatmigen 144  
MODE  
  communication paradigm 123, 124, 126  
MSCF 168  
multiple conversations 54

### N

- name
  - data structure 126
- name length for partner names
  - CPI-C 53
- notational conventions 26
- number of sessions/associations (XATMI) 131

### O

- OCTET STRING 112
- openUTM client application 33
- OSI-TP association 42
- outbound conversation 38, 54
- outgoing conversation 38
- overhead, buffer 121

### P

- parallel jobs 108
- PCMX 13
- PGWT
  - TAC 131
  - TAC classes 131
- process switching
  - service 121
- PROG 125
- program interface
  - CPI-C 37
  - TX 150
  - XATMI 114
- program name
  - service 125
- programming languages
  - CPI-C 95
  - TX 149
- programming rules
  - CPI-C programs 83
- protocol 28
- protocols
  - XATMI 103

### R

- Readme files 15
- Receive (conversation state) 64
- Red Hat 9

- reference model (DTP) 28
- remote-service-name 123
- REQP 124
- request 103
- request offered 124
- request with no response 109
- request-response 124, 126
- requester 103
- Reset (conversation state) 64
- return value 118
- revent 118

### S

- sample application, CPI-C 44
- Send (conversation state) 64
- send and receive mode 53
- send control 53
- Send-Pending (conversation state) 64
- server side of conversation see acceptor
- server/server link 32
- server/server network 37
- service
  - call 117
  - data transfer 117
  - initialize 117
  - XATMI 103
- service info structure 117
- service offered 124
- session 38, 42
  - characteristics 47
  - generation specifications 98
  - maximum number 55
- sessions (XATMI) 131
- side information 47
- single request 108, 109, 124, 126
- Solaris 9
- standalone UTM application 8
- start
  - XATMI application 137
- subtypes 111
- SUSE 9
- SVCP 124
- SVCU 122
- sync pointing 56

sync\_level, CPI-C 56  
synchronization of processing  
  CPI-C 56  
synchronous request-response paradigm 107  
syntax  
  LCF definition file 122

## T

T.61 character set 148  
T.61 string 112  
TAC  
  XATMI service 117

## tasks

  number 131  
TM (Transaction Manager) 28  
tpacall 107  
tpcall 107  
tpconnect 109  
tpdiscon 110  
TPEEVENT 118  
tperrno 118  
tpgetrply 108  
tprecv 109  
tpsend 109  
tpservice 118  
TPSVCSTART 118

## trace

  CPI-C program 100  
  TX program 157  
  XATMI program 141  
trace file  
  name in TX programs 158  
  name in XATMI programs 142

## transaction code

  for CPI-C program unit 97  
  XATMI 123, 125  
Transaction Manager (TM) 28  
transaction\_control characteristic 149

## transition 64

## TX

  Integration in openUTM 31  
typed buffer 104, 110, 119  
  rules 119  
  types 111

typed record 111

## types

  XATMI 111

## U

UPIC protocol 28, 31  
usage, typed buffer 119  
user buffer 111  
user data, convert 57  
UTM client application 33  
UTM cluster application 8  
  cluster administration journal 164

## W

Windows system 9

## X

X\_C\_TYPE 111, 112  
  conversion 113  
X\_COMMON 111, 112  
  conversion 113  
X\_OCTET 111  
  BUFFERS operand 124  
XATMI 103  
  client/server link 33  
  Integration in openUTM 31  
  KDCDEF generation 131  
  messages 144  
  program interface 114  
  server/server link 32  
XATMI program  
  error diagnostics 141  
XATMI U-ASE 104, 111  
XATMI-TRACE 141  
XATMIGEN 128  
xatmigen  
  messages 144

