English

FUJITSU Software

openUTM-Client V4.0 for the OpenCPIC Carrier System

Client Server Communication with openUTM

User Guide

Edition February 2017

Comments… Suggestions… Corrections…

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

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

Certified documentation according to DIN EN ISO 9001:2008

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

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

Copyright and Trademarks

Contents

1 Preface

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

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

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

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

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

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

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

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

1.1 Brief description of the product

The product “openUTM-Client V4.0, carrier system OpenCPIC” (in short: OpenCPIC) allows client-server communication and transaction-oriented processing in distributed systems on the basis of the X/Open DTP reference .

In this manual, the abbreviation OpenCPIC is always used as the product description.

OpenCPIC allows implementation of the following program interfaces:

– X/Open CPI-C® (Common Programming Interface for Communication) Version 2.0

– X/Open XATMI® (X/Open Application Transaction Manager Interface)

– X/Open TX® (Transaction Demarcation Interface)

Implementation is based on the OSI-TP (Open Systems Interconnection Distributed Transaction Processing) protocol.

OpenCPIC allows the production and operation of transaction applications in distributed systems, and supports transaction-oriented security, consistency of user data during access to databases, as well as restarting.

1.2 Purpose and target group of this manual

This manual is addressed both to the system administrator who carries out the necessary preparations and implements OpenCPIC, and to the developer of application programs.

Familiarity with Unix and Linux systems and CMX is a prerequisite here. For configuring the applications, a fundamental knowledge of the OSI-TP communication model is recommended.

As a developer of application programs for OpenCPIC, you should be familiar with the X/Open CPI-C V2.0 and X/Open XATMI interfaces. Transaction-oriented operations also require a knowledge of the X/Open TX interface.

For database access, you need to be familiar with the corresponding program interface of your database system (e.g. Informix-SQL).

In general, an understanding of this document is facilitated by a knowledge of the X/Open DTP reference model. Chapter 2 provides a brief introduction which is restricted to fundamental aspects of the model and the embedding of OpenCPIC.

The literature index at the end of the manual provides a list of all X/Open documents relevant to OpenCPIC.

1.3 Summary of contents of openUTM manuals

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

1.3.1 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“ in the **Search by product** field to display all openUTM manuals of the corresponding version.

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

Introduction and overview

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

Programming

- You will require the **Programming Applications with KDCS for COBOL, C and C++** manual to create server applications via the KDCS interface. This manual describes the KDCS interface as used for COBOL, C and C++. This interface provides the basic functions of the universal transaction monitor, as well as the calls for distributed processing. The manual also describes interaction with databases.

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

Configuration

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

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

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

Linking, starting and using UTM applications

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

Administering applications and changing configurations dynamically

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

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

Testing and diagnosing errors

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

Creating openUTM clients

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

- The **openUTM-Client for the UPIC Carrier System** describes the creation and operation of client applications based on UPIC. The manual basicly comprises the description of the CPI-C and XATMI interfaces..

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

Communicating with the IBM world

If you want to communicate with IBM transaction systems, then you will also require the manual **Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications**. This describes the CICS commands, IMS macros and UTM calls that are required to link UTM applications to CICS and IMS applications. The link capabilities are described using detailed configuration and generation examples. The manual also describes communication 6via 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

1.3.2 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.

1.3.3 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*.

1.4 Notational conventions

The following typographic notational conventions are used in this manual:

	This symbol indicates important information which must be read.
Italics	This indicates commands, variables, constants, path names as well as file and program names in plain text.
`typewriter font`	This indicates messages and screen outputs as well as examples of programs and data entries.
Semi-bold base type	In syntax display, this font indicates syntax elements which must be entered as they are displayed.
Normal base type	In syntax display, this font indicates variable syntax elements. These elements (or placeholders) represent other characters or character strings which must be entered in their place.
[ ]	In syntax display, square brackets indicate optional values. These brackets are not syntax elements and must not be entered. In plain text display, numbers enclosed in square brackets refer to literature listed at the end of this manual.
{}	In syntax display, curly brackets contain alternative parameters or values. Only one value or parameter must be enclosed in a pair of curly brackets.
|	In syntax display, a vertical line separates alternative entries, one of which must be selected.
openCPIC-path	Designates the installation directory of OpenCPIC.

2 Introduction to the OpenCPIC product

After a brief introduction to the topic of distributed transaction processing, this chapter describes the embedding of OpenCPIC in the X/Open DTP reference model. It also provides an overview of the structure of OpenCPIC and the interaction of the various components.

2.1 Distributed processing with global transaction management

A transaction consists of a sequence of actions required for executing a firmly outlined, application-specific unit of work. For example, a transfer between two accounts - consisting of a debit to one account and a corresponding credit to the other - constitutes a transaction.

Transactions are identified by four fundamental attributes, also termed ACID criteria (Atomicity, Consistency, Isolation, Durability):

– Atomicity
  A transaction is either performed completely or not at all.

– Consistency
  On cancellation (rollback) of a transaction, all the involved resources are returned to their original state.

– Isolation
  The results of the involved actions remain externally invisible until the transaction has been completed successfully (committed).

– Durability
  The results of a transaction are durable.

On ending a transaction, all the executed actions are either confirmed (committed) or reset (rolled back). A rollback is required if at least one of the actions could not be performed or led to an error.

During distributed processing with global transaction management, the programs and resources involved in a transaction (e.g. databases) are distributed among various systems in a network.

In the case of global transactions, it is necessary to coordinate the interactions between the distributed resources and programs involved in the transaction. This coordination is performed using a two-phase commit protocol which ensures the consistency of all decisions made.

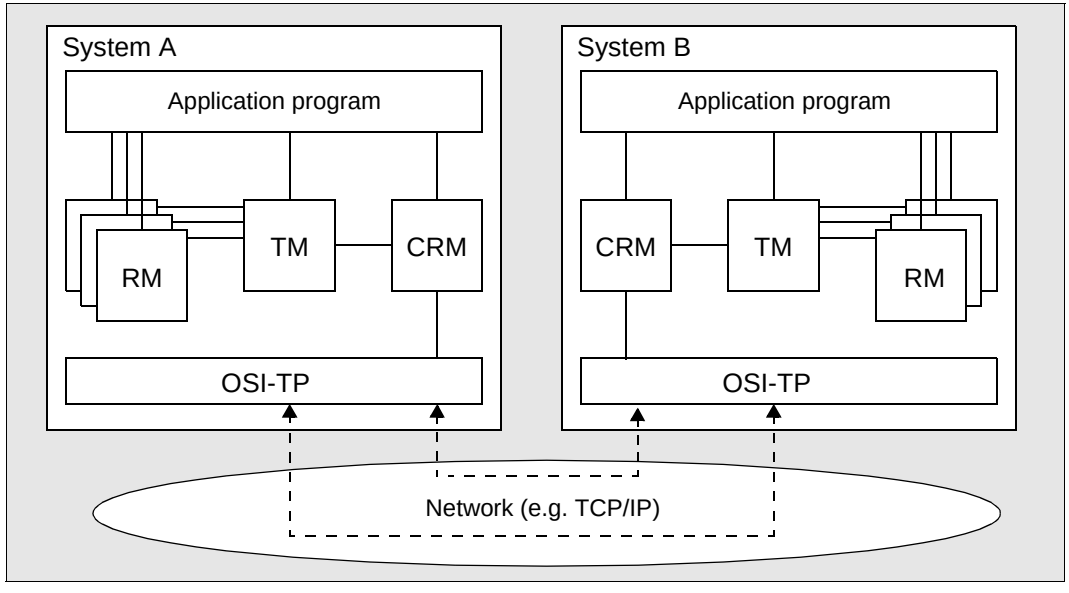
2.2 The OpenCPIC and X/Open DTP reference models

The steadily growing number of networked computer systems requires an ever increasing compatibility between software components developed by different manufacturers and running on different computer types with different operating systems. This is made possible by open systems. Open systems ensure the proper interaction and portability of applications and data in heterogeneous system environments by incorporating international standards for interfaces and data formats.

For distributed processing in open systems, X/Open has developed a reference model: the distributed transaction processing reference model V2/11.93.

The X/Open DTP reference model uses various standardized interfaces for communication between client and server applications as well as distributed processing.

The following diagram shows the components of the DTP reference model.



X/Open DTP reference model

Application program
: The two application programs are the end points of any program-to-program communications process. An application program or transaction program is, for example, a server or client in a distributed processing system.

CRM
: Communication between the application programs is controlled by the communication resource managers (CRM). The CRMs on different systems communicate via the OSI-TP transaction protocol which is defined by ISO and can be based on

various network protocols such as TCP/IP or X.25.
CPI-C, XATMI and TxRPC were defined by X/Open as interfaces between the application programs and CRM.

RM As application programs communicate with remote applications via CRMs, they access local databases via local resource managers (RM). The interfaces used here are specified by the respective RMs. X/Open defines various interfaces between application programs and RMs, e.g. SQL and ISAM.

TM The transaction manager (TM) monitors the execution of secured transactions. It controls the start and end of transactions. The two-phase commit protocol is used for coordination with the local RM and remote TM. This ensures the consistency of data modifications for all the involved resource managers. The TM also performs recoveries after errors have occurred.
X/Open defines the XA interface for communication between the TM and RM, and the TX (transaction demarcation) interface for communication between the AP and TM. The XA+ interface was defined for communication between the CRM and TM.

OSI-TP
The CRM and TM on different systems communicate via the OSI-TP (open systems interconnection - transaction processing) protocol which is defined by ISO and can be based on various network protocols such as TCP/IP or X.25. OSI-TP is the protocol for the uppermost layer (application layer) of the OSI protocol stack.
X/Open has defined the XAP-TP interface for standardized access to OSI-TP by applications. XAP-TP is an extension of the X/Open interface XAP allowing access to the ACSE and presentation layers of the OSI protocol stack.

The OpenCPIC product implements the following components of the X/Open DTP reference model:

– Communication resource manager (CRM)
– Transaction manager (TM)
– CPI-C interface between the application program and CRM
– XATMI interface between the application program and CRM
– TX interface between the application program and TM
– XAP-TP interface between the CRM and OSI-TP

In OpenCPIC, communications between the CRM and TM take place via internal interfaces instead of XA+.

The terms “application” and “application program” described in the following carry particular significance:

Application

All local application programs on a system constitute, together with the local communication resource manager (CRM) and local transaction manger (TM), an application. In OSI (open systems interconnection) terminology, the term application entity (AE) is also used instead of application. An application in the network is addressed via a transport address and an application entity title (AE title). The transport address and AE title of an application must be unique throughout the network.
In the case of OpenCPIC, the OpenCPIC manager on a system constitutes, together with the local application programs, an application. Some typical applications, in addition to OpenCPIC, are UTM and CICS applications.

Application program

An application program is part of an application. Equivalent terms here are transaction program (TP, used particularly in conjunction with CPI-C), program unit (with openUTM) and transaction processing service user (TPSU, with OSI-TP). This manual always uses the term “application program”.
An application program is identified by means of a local name. This name is only unique within an application. For unique addressing of an application program in the network, the transport address is thus also required.
Equivalent terms are transaction program name (TP name) with CPI-C, transaction code with openUTM, and TPSU title with OSI-TP
As in OpenCPIC, application programs can be independent processes engaging in inter-process communication (IPC) with the CRM and TM. As in the case of openUTM however, they can also be linked with the CRM, TM and other application programs to form a single application (see above).

2.3 Areas of usage of CPI-C, XATMI and TX

The following table shows the areas in which these interfaces can be suitably used:

Interface	Areas of usage
CPI-C	This is an interface for program-to-program communication. It is suitable for connecting presentation clients (PC, workstation) to UTM applications and for communication between server applications. As CPI-C is also defined as part of IBM SAA, it is particularly suitable for applications intended for use in IBM environments.
XATMI	This is an interface for program-to-program communications. AS XATMI is also supported by other transaction monitors, it is particularly suitable for applications intended to communicate with those of other transaction monitors, such as TUXEDO.
TX	This is an interface between the program and transaction monitor, and intended to control global transactions. It is particularly suitable for coupling OpenCPIC applications with UTM applications.

2.4 Components of OpenCPIC

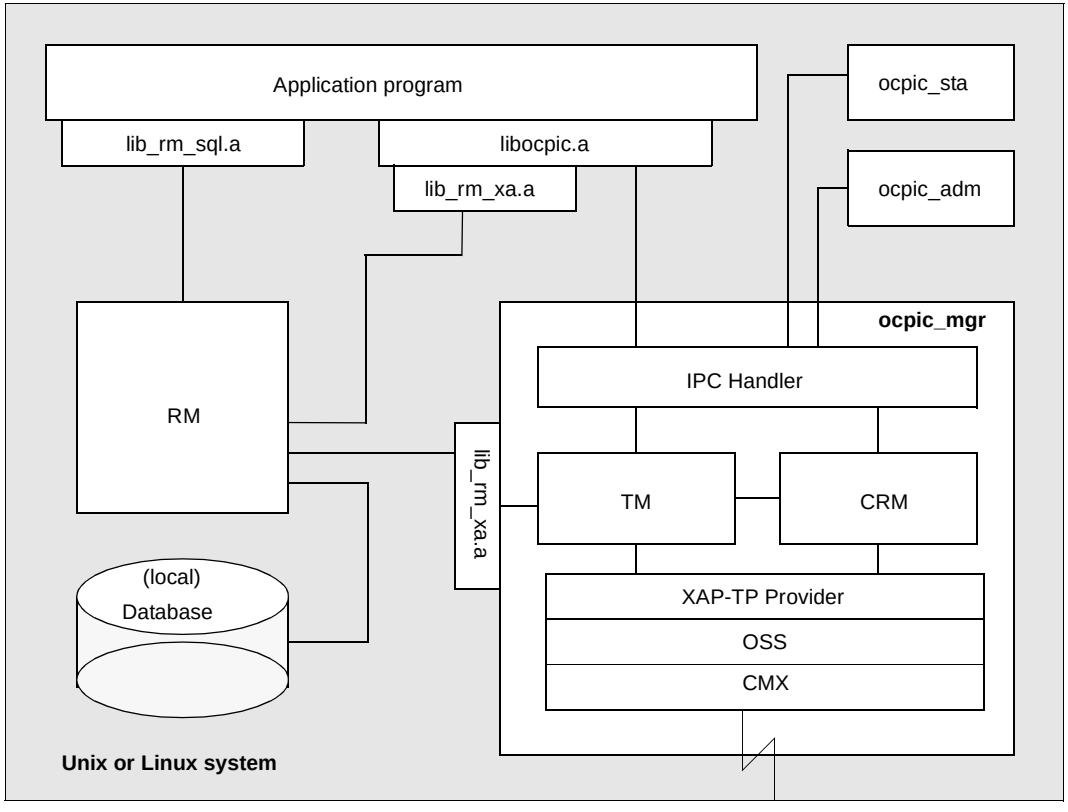
The product OpenCPIC consists of the following components:

– The program *ocpic_mgr*, termed OpenCPIC manager in the following.
  As the central manager process, this program acts as the communication resource manager (CRM) and transaction manager (TM) for all the application programs of a local application.

– The library *libocpic.a*, in which the interface functions of CPI-C and TX are implemented. *libocpic.a* is a static library which must be linked with each OpenCPIC application program.

– Utility programs e.g. for generation (*ocpic_gen*) and administration (*ocpic_adm*, *ocpic_sta*)

The following diagram shows a simple OpenCPIC application in a Unix or Linux system, consisting of the OpenCPIC manager (*ocpic_mgr*) and a local application program which has access to a local database via a local resource manager (RM). Access to a distributed database is also possible, if the resource manager supports this. The diagram shows the structure of an OpenCPIC application and the interaction of the individual components of OpenCPIC on a local system.



Components of an OpenCPIC application

The program *ocpic_mgr* is the central manager process of the application. It implements the CRM and TM components of the X/Open DTP reference model. Inter-process communication between the OpenCPIC manager and the local application programs as well as the administration programs (*ocpic_sta*, *ocpic_adm*) takes place via named pipes and is controlled by the IPC handler in the OpenCPIC manager. Like the administration programs, the CPI-C and TX functions of the library *libocpic.a* transmit and receive messages via special, named pipes. The IPC handler forwards the messages to the responsible component (CRM or TM) and relays the corresponding responses back.

The CPI-C and TX functions are converted to the OSI-TP protocol by a component named XAP-TP component. The XAP-TP component transmits the OSI-TP protocol elements via the network to the partner application and forwards data received from the partner to the CRM and TM. For this purpose, it is based on the communication product CMX.

The application program in this example communicates with the local resource manager via an SQL interface. This interface is provided by the local database system in the form of an interface library named *lib_rm_sql.a*.
The XA interface is implemented in the library *lib_rm_xa.a*, which is also a component of the locally installed database product. As the TX functions of the library *libocpic.a* and the transaction manager invoke XA functions, both the application program and the program *ocpic_mgr* must be linked with this library.

2.5 Important terms

Certain important terms used frequently in this manual are explained in the following:

Association

Associations are links between application entities, i.e. layer-7 links in the ISO reference model. For any dialog between two application programs, an association must first be established between the two partner applications.

Dialog and conversation

A dialog in OSI-TP is a logical connection between two TPSU (transaction processing service users).
A conversation is a logical connection between two CPI-C application programs. For any conversation between two application programs in OpenCPIC, an OSI-TP dialog is opened by the corresponding partner applications. In other words, each conversation is assigned exactly one dialog and vice versa.

Superior and subordinate

The superior or initiator of a conversation is the partner who opens the conversation with *Allocate (CMALLC)*. The subordinate or acceptor is the partner who accepts the conversation with *Accept (CMACCP)* or *Accept_Incoming (CMACCI)*. On the OSI-TP level, the superior in a dialogue is correspondingly the partner who opens the dialog with a *TP-BEGIN-DIALOGUE-request* and the subordinate is the partner who receives this request in the form of a *TP-BEGIN-DIALOGUE-indication*.

Root, intermediate and leaf nodes of a transaction

In OSI-TP, distributed transactions are organized in the form of trees (refer to [32]). The involved TPSU (transaction processing service user), i. e. the application programs holding the dialogues and conversations form the nodes of the transaction tree (transaction nodes). The root node of the transaction tree is the only node with no superior. Each node can have a maximum of one superior, but several subodinates. A node possessing a superior and at least one subordinate is termed intermediate node. Nodes with no subordinate are termed leaf nodes.

Protected conversations

Protected conversations are those currently involved in an ongoing transaction. Consequently, the dialogue underlying the conversation is a part of the transaction tree. In OSI-TP such a dialogue is termed as “Coordinated” (refer to [32]). The application program holding the protected conversation forms a node in the transaction tree.

Precommit phase, commit phase

These are the two phases of the two-phase commit protocol. This protocol procedure is controlled by the involved transaction managers.

In the precommit phase, all the involved transaction nodes are requested to put their respective local resources into a state allowing successful completion of the transaction (“prepare to commit”). This is done by sending a corresponding CCR protocol element from the root node through the entire transaction tree. Once all nodes have indicated their readiness to complete the transaction (“ready to commit”), they are requested by the root node to modify their local resources. If, during this phase, any node indicates that its local resources are not ready to commit, or a fault occurs, e.g. failure of a transaction node, then all transaction nodes are requested to perform a rollback, i.e. restore their local resources to the initial state.

The decision (commit or rollback) made in the precommit phase is implemented in the commit phase. If a decision to commit was made, then the transaction is completed successfully, after which modifications made to the involved resources can no longer be undone. If a decision to rollback was made, then all involved resources are restored to their original state.

Heuristic decisions

Under certain circumstances (e.g. failure of the transaction manager) local resource managers can make heuristic decisions during the commit phase. This means that a local resource manager decides - independently of the transaction manager - to end the local transaction branch with a commit or rollback. When the transaction manager informs the local resource manager of its decision at a later stage, the resource manager replies that it has already terminated the local transaction branch. This can lead to inconsistencies in the transaction tree, if the heuristic decision of a local resource manager does not conform to the decision made by the transaction managers. This situation is termed “heuristic mix condition” or “heuristic hazard condition”. A heuristic outcome is said to be involved here. Measures for remedying such inconsistencies are to be implemented by the system administrator, in accordance with the resource manager product used.

3 Generating OpenCPIC

For starting, the OpenCPIC manager requires certain configuration details which the system administrator must supply in the form of a configuration file. This file is created in two steps:

1. Creation of a generation file with a text editor

   All the required information is entered in readable form (ASCII text) in the generation file. The structure of this file is described in section “Structure of the generation file” on page 30

2. Creation of the binary configuration file *conffile* from the generation file

   The configuration information must be available to the OpenCPIC manager in a particular, internal format. For this, the generation file is prepared with the command *ocpic_gen*. This command is described in section “Starting generation programs” on page 49.

3.1 Structure of the generation file

The generation file consists of a sequence of statements, each of them having the following general format:

statement-type name [,parameter$_1$[,..., parameter$_n$]]

statement-type
: indicates the type of statement. The following statement types can be used:

  – LOCAPPL (see page 32)
  – PARTAPPL (see page 34)
  – LOCAPRO (see page 39)
  – SYMDEST (see page 42)
  – APPLICATION-CONTEXT (see page 47)

name
: is the name of the objects to be generated with this statement. The name must be located in the same line as the statement type. It is freely selectable and must only occur once in the generation file.

parameter
: consists of a keyword, the character '=' and a value.

  Example: **ASSOCIATIONS = 2**

  Blank value allocations (e.g. **ASSOCIATIONS =**) are not permissible.

  If a statement contains several parameters, they must each be separated by commas. The last parameter of a statement must not be followed by a comma.

  The parameters and related value ranges permissible for each statement are described in separate sections. Parameters in square brackets are optional. Missing optional parameters are replaced with default values.

The following rules apply:

- A generation file must contain exactly one LOCAPPL statement and at least one PARTAPPL statement. The remaining statements are optional.
- The sequence of statements in the generation file as well as the sequence of parameters in a statement can be freely selected.
- Every statement must begin on a new line and can be several lines long. However, parameters must not be longer than one line.
- Lines beginning with an asterisk (*) or number sign (#) are treated as comments and ignored by *ocpic_gen*.

General syntax rules for names

Permissible characters for names are all printable ASCII characters (letters, numbers, symbols) except for the following special characters:

– Asterisk (*)
– Comma (,)
– Semicolon (;)
– Blank character (0x20)
– Tabulator (0x09)

General syntax rules for object identifiers

An object identifier is a special ASN.1 data type defined in ISO 8824 [29]. Object identifiers generally consist of a list of elements which can be entered as integers or names.

In OpenCPIC, object identifiers must always be specified as a list of at least two and at most ten non-negative integers separated by commas. This list is to be enclosed in parentheses.

Example: **(1, 5 ,6）**

Note that the number of elements in the list is of relevance. For example, (1, 5, 6) and (1, 5, 6, 0) are two different object identifiers.

The following value ranges apply to the elements of an object identifier:

1st element: 0 - 2 (refer to the table on page 31)

2nd element: If the first element is 0 or 1: 0 - 39,
otherwise 0 - 67108863

3rd up to the 10th element: 0 - 67108863

Object identifiers must be registered by an authorized agency. The first element of an object identifier indicates the agency authorized to register the object identifier (i.e. the following elements). The allocations applicable in this context are listed below (also refer to ISO 8824 [29], Annex B):

Value of the first element	Authorized registration agency
0 (ccitt)	CCITT
1 (iso)	ISO
2 (joint-iso-ccitt)	CCITT and ISO

3.1.1 LOCAPPL - Defining the local application

Syntax

LOCAPPL	locappl_name**,**
APT	**=** apt_objid**,**
AEQ	**=** aeq_num

Description

Use this statement to define your local application, i.e. the OpenCPIC manager on the local system, as the local access point for communication via the OSI-TP protocol. All CPI-C conversations to be held by your local application programs are controlled via this one access point. Consequently, a generation file must contain exactly one LOCAPPL statement.

locappl_name

This is the name with which the OpenCPIC logs into CMX. It must be generated in TNSX (transport name service in the Unix or Linux system) as *global name*. How to make entries in TNSX is described in the CMX user manual [22].

The *locappl_name* consists of 5 name sections separated by points:

NP5.NP4.NP3.NP2.NP1

If name sections are empty, then the points at the end of the name can be omitted. The maximum lengths of the individual name sections are:

NP1	2	characters
NP2	16	characters
NP3	16	characters
NP4	10	characters
NP5	30	characters

More details on the syntax of the *global name* are provided in the CMX user manual [23] (Description of the program call *t_getaddr*).

Note: In addition to a T selector for layer 4, a P and an S selector must also be entered for layers 5 and 6 in TNSX. It is advisable to set blank entries for S and P selectors (also refer to the CMX user manual [22]).

apt_objid

This specifies the application process title (APT) in the form of an object identifier (refer to page 31). The application process title, together with the application entity qualifier (AEQ), forms the application entity title (AET) of the local application.
As described on page 31, object identifiers need to be registered by an authorized agency. This also applies to the application process title, particularly if you work in an open network.
In a private, closed network, the network operator is personally responsible for the uniqueness of the application process title; for this reason, official registration is not mandatory here.

aeq_num

This is an integer between 1 and 67108863 specifying the application entity qualifier (AEQ). It is freely selectable and forms, together with the application process title, (APT), the application entity title (AET) of the local application.

Remarks

1. APT = *apt_name* as well as AEQ = *aeq_name* must be specified.
2. The application entity title (AET) formed from APT and AEQ must allow unique addressing of the local OpenCPIC manager in the network. The AET is specified as a parameter in the OSI-TP protocol element *TP-BEGIN-DIALOGUE-request*.

3.1.2 PARTAPPL - Defining the partner application

Syntax

PARTAPPL	partappl_name**,**
APT	**=** apt_objid**,**
AEQ	**=** aeq_num [**,**
APPL-CONTEXT	**=** context_name][**,**
CCR	**=** ccr_option][**,**
ASSOCIATIONS	**=** associations_number][**,**
CONNECT	**=** connect_number][**,**
CONTWIN	**=** (contwin_min contwin_max)][**,**
IDLETIME	**=** idle_number]

Description

This statement is used to generate a partner application, e.g. an OpenCPIC manager or UTM application on a remote system. Every partner application to be accessed in the network must be generated with a separate PARTAPPL statement. It is not possible to establish links with non-generated partner applications. The PARTAPPL statement is thus also an aid for data protection.

partappl_name

This is the name of the partner application in the remote system. The same syntax rules apply here as for the *locappl_name* in the LOCAPPL statement (refer to page 32).

The *partappl_name* must be generated in TNSX as *global name*. How to make entries in TNSX is described in the CMX user manual [22].

Note: In addition to a T selector for layer 4, a P and an S selector must also be entered for layers 5 and 6 in TNSX. It is advisable to set blank entries for S and P selectors (also refer to the CMX user manual [22]).

apt_objid

This specifies the application process title (APT) in the form of an object identifier (refer to page 31). The application process title, together with the application entity qualifier (AEQ), forms the application entity title (AET) of the partner application.
As described on page 31, object identifiers need to be registered by an authorized agency. This also applies to the application process title, particularly if you work in an open network.
In a private, closed network, the network operator is personally responsible for the uniqueness of the application process title; for this reason, official registration is not mandatory here.

aeq_num

This is an integer between 1 and 67108863 specifying the application entity qualifier (AEQ). It forms, together with the application process title (APT), the application entity title (AET) of the partner application.

context_name

This is the name of the application context agreed for linkage with the partner application. Here, you can either specify one of the predefined application context names described in the following, or enter the name of an application context defined by you in the generation file (refer to page 47, APPLICATION-CONTEXT statement). If a user-defined application context is selected, the APPLICATION-CONTEXT statement must be located before the PARTAPPL statement in the generation file.

The following application context names are predefined:

= def-cont Standard application context, consisting of the application service elements ACSE, TP-ASE, CCR (optional, refer to the parameter CCR = *ccr_option*), and UDT. UDT (with octet string mapping) is the abstract syntax; BER (basic encoding rules) is used as the transfer-syntax (default value).

= utm-secu This application context must be used if the partner is a UTM application and the CPI-C calls *Set_Conversation_Security_Type (CMSCST)*, *Set_Conversation_Security_User_ID (CMSCSU)* and *Set_Conversation_Security_Password (CMSCSP)* are to be used (refer to note 2 on page 38).

= xatmi This application context must be used if both partner programs use the X/Open interface XATMI.

The following overview shows the object identifiers of the predefined application context names as well as the abstract syntaxes involved in each case. BER (basic encoding rules) is always used as the transfer syntax.

Application context name (keyword)	Object identifier	Abstract syntaxes involved
def-cont	(1, 0, 10026, 6, 2)	UDT, possibly CCR
utm-secu	(1, 3, 12, 2, 1107, 1, 6, 1, 3, 0)	UDT, UTM security, CCR
xatmi	(1, 2, 826, 0, 1050, 4, 2, 1)	XATMI, possibly CCR

The following table contains the object identifiers of the abstract and transfer syntaxes used. .

Syntax name	Object identifier
BER	(2, 1, 1)
CCR	(2, 7, 1, 1, 1)
UDT	(1, 0, 10026, 6, 1, 1)
UTM security	(1, 3, 12, 2, 1107, 1, 6, 1, 2, 0)
XATMI	(1, 2, 826, 0, 1050, 4, 1, 0)

Further information is provided in the section “APPLICATION-CONTEXT - Defining an application context” on page 47.

ccr_option

This indicates whether the partner application supports the abstract syntax CCR (commitment, concurrency and recovery), i.e. whether the application context of the partner application contains CCR syntax. CCR syntax is required for holding conversations with *sync_level CM_SYNC_POINT* and *CM_SYNC_POINT_NO_CONFIRM*. This parameter is optional and can assume the following values:

= **YES** The selected application context contains CCR syntax (default value).

= **NO** The selected application context does not contain CCR syntax.

For the application context name *utm-secu*, this parameter is not evaluated. The application context *utm-secu* contains CCR syntax by default.

associations_number

This is the maximum number of parallel associations between the local application and the partner application. Permissible values for *associations_number* are integers from 1 to 256. The default value is 1.

Associations are layer-7 links in the ISO reference model. At present, they are mapped to transport links on a one-to-one basis. For this reason, it is advisable to use transport systems which support parallel transport links, e.g. TCP/IP and OSI protocols. It is pointless to assign *associations_number* a value larger than the maximum number of parallel transport links supported by the transport system.

An association needs to be set up for every dialog between two application programs (and, thus, for every CPI-C conversation). For this reason, *associations_number* is also the upper limit for the maximum number of dialogs and conversations which can be held simultaneously between the application programs of the local and partner applications.

connect_number

This is the number of associations for the partner application, which are to be set up automatically on start of the local OpenCPIC manager. Permissible values are integers from 0 to the *associations_number*. The default value is 0.

If the OpenCPIC manager cannot set up the required number of associations on starting, a new attempt is made every five minutes.

contwin_min, contwin_max

These are the minimum and maximum number of associations for which the local OpenCPIC manager is the contention winner. Permissible values are integers from 0 to the *associations_number*. *contwin_min* must be smaller than or equal to *contwin_max*. The default value for *contwin_min* is 0, the default value for *contwin_max* is *associations_number*.

The contention winner is responsible for managing the association. Irrespective of which side is the contention winner for an association, both partner applications can initiate a dialog or conversation on this association. If a collision occurs, i.e. if both partners attempt to initiate a dialog or conversation simultaneously on the association, then the association is reserved by the contention winner, while dialog setup is rejected on the side of the contention loser.

It is advisable to configure contention winner associations only if a dialog is to be actually set up from the local side (i.e. your local CPI-C application programs establish active conversations with *Allocate (CMALLC)*). If no dialogs are set up on the local side (i.e your application programs only accept requests for dialog setup from the partner side using *Accept_Conversation(CMACCP)* or *Accept_Incoming (CMACCI)*), then you should set *contwin_min* and *contwin_max* to 0.

If the local side is to set up active dialogs and accept dialog requests by the partner, then it is advisable to proceed as follows: set *contwin_min* to the number of (simultaneously active) dialogs to be set up by the local application and *contwin_max* to a value smaller than *associations_number*, so that the difference *L = associations_number - contwin_max* is larger than or equal to the number of (simultaneously active) dialogs which are set up by the partner side. As *L* is larger than or equal to the number of contention loser associations, this ensures that all incoming dialog requests are accepted by the local application.

Note: If the partner is a UTM application, *contwin_min* and *contwin_max* must be matched with UTM generation on the partner side.

idle_number

This is the monitoring time (in seconds) for the idle state of the associations for the partner application. *idle_number* is entered as an integer from 0 to 32767. The default value is 0.

If an association is not reserved by a dialog in the specified time (i.e. longer than *idle_time* seconds), the OpenCPIC manager clears the association. A value of *idle_number = 0* means that the idle state of the association is not monitored.

Notes

1. APT = *apt_name* and AEQ = *aeq_name* must be specified.
2. The application entity title (AET) formed from APT and AEQ must allow unique addressing of the partner application in the network. The AET is specified as a parameter in the OSI-TP protocol element *TP-BEGIN-DIALOGUE-request*.
3. If *partappl_name* is to be used as an argument for the CPI-C call *Set_Partner_LU_Name (CMSPLN)* or extracted with *Extract_Partner_LU_Name (CMEPLN)* (refer to the note 1 on page 44 of the SYMDEST statement) then its total length must not exceed 17 characters, due to the definition of the CPI-C interface. Otherwise *CMSPLN* returns the value *CM_PROGRAM_PARAMETER_CHECK*, while *CMEPLN* truncates the name after 17 characters.

3.1.3 LOCAPRO - Defining a local OpenCPIC application program

Syntax

LOCAPRO	locapro_name**,**
PATH	**=** path_name**,**
LOGIN	**=** login_name

Description

This statement is used to define a local OpenCPIC application program (**Loc**al **A**pplication **Pro**gram).

Every local application program (AP) which is to be started automatically by the local OpenCPIC manager due to a dialog request received from the partner application, must be defined with a LOCAPRO statement in the generation file.

Local application programs which are not to be started by the OpenCPIC manager need not be configured. Such APs need to log their names into the OpenCPIC manager with the CPI-C call *Specify_Local_TP_Name (CMSLTP)* before incoming dialog requests can be accepted.

locapro_name

This is the local name of the AP as specified by the partner application during dialog establishment (parameter in the OSI-TP protocol element *TP-BEGIN-DIALOGUE-request*).

locapro_name is normally not the name of an executable file in the Unix or Linux system, but represents a symbolic name linked with a program name via the parameter PATH = *path_name* (see below).

This symbolic name corresponds to TPSU title in OSI-TP terminology, and to TP name in CPI-C terminology.

locapro_name can be up to 64 characters long. Only characters from the character set *T61String* are permissible (refer to the “section “Character sets” on page 191”)

Note: If the local AP of a UTM partner application is to be addressed, then *locapro_name* must not be longer than eight characters, and must only consist of upper-case letters and numbers.

path_name

This is the (fully qualified) Unix or Linux path name of the local application program to be started on reception of a dialog request for the symbolic name *locapro_name* (see above). The *path_name* can be up to 64 characters long and must start with '/'.

login_name

This is the Unix or Linux user ID under which the local application program is to be started. After having been started by the local OpenCPIC manager, the local AP is run in the environment with the access rights of this user ID. The *login_name* has a maximum length of eight characters.

Notes

1. On the partner side, the local name corresponds to the *TP_Name* in the side information or the name set by the partner AP using *Set_TP_Name (CMSTPN)*. If the partner is a UTM application, then the local name corresponds to the value of the parameter RTAC in the LTAC statement of the UTM generation (KDCDEF). A sample configuration is provided in the chapter "Communicating with partner applications" on page 117.
2. Every local application program which wants to accept dialog requests made by a partner AP using *Accept_Conversation* (*CMACCP)* or *Accept_Incoming* (*CMACCI)* must be logged into the OpenCPIC manager with at least one local name. Login is performed either on automatic program start by the OpenCPIC manager - in which case the local name must be generated with the LOCAPRO statement described here - or with the CPI-C call *Specify_Local_TP_Name (CMSLTP)* during program execution.
   Programs which have been generated in a LOCAPRO statement but are started via an operator command instead of the OpenCPIC manager are not logged in automatically with the local name generated in the LOCAPRO statement. Every AP started by an operator command must login itself using *CMSLTP*.
3. When the local OpenCPIC manager receives a dialog request from a partner application for a particular local name, the validity of this name is checked in the following sequence:
   – If at least one active AP has been logged into the OpenCPIC manager with the relevant local name, then the dialog request is placed in a queue until one of these APs accepts the dialog with *Accept_Conversation (CMACCP)* or *Accept_Incoming (CMACCI)*. If, on arrival of the dialog request from the partner, an AP is already waiting for a response to an outstanding *CMACCP/CMACCI* call, the dialog request is forwarded directly to the local AP and the conversation is set up on the local side. If several APs with the same local name want to accept a dialog request, the sequence in which incoming dialog requests are forwarded need not be identical to the sequence in which the *CMACCP/CMACCI* calls are issued.

Users of OpenCPIC V2.0 should note that incoming dialog requests are now handled differently. As dialog requests are placed in a queue if at least one application program with the specified local name is active, certain measures must be taken before several application programs with the same name can be started automatically at the same time. If you want to perform a multiple automatic start of the same application program, proceed in one of the two following ways:

a) Define several different local names for the application program (every local application program can have any number of local names).

b) Declare the local names as invalid after successful *Accept_Conversation (CMACCP)* or *Accept_Incoming (CMACCI)* via *Release_Local_TP_Name (CMRLTP)* (also refer to note 4).

– If no active AP of this name has yet logged into the OpenCPIC manager although a corresponding LOCAPRO statement exists in the current configuration, the OpenCPIC manager tries to start the corresponding program (parameter PATH = *path_name*) in the local system. The dialog request is placed in a queue until a local AP accepts a dialog for the relevant local name with *CMACCP/CMACCI*. (This need not be the program started by the OpenCPIC).

– If, on reception of a dialog request, no local AP has logged in with the required name and no local AP of this name was generated with a LOCAPRO statement, the local OpenCPIC manager rejects the dialog request, which leads to cancellation of the dialog on the partner side. In the case of a CPI-C partner program, for example, this generates the return value *CM_DEALLOCATED_ABEND*.

– Dialog requests are also rejected if the start of the local AP fails or the interruption of a local AP invalidates the required local name and no other generated AP of this name can be started.

4. CPI-C allows application programs which have logged into the local system under a special local name via the call *Specify_Local_TP_Name (CMSLTP)* to log out again under the same name. This is done via the call *Release_Local_TP_Name (CMRLTP)*. A local name generated with the LOCAPRO statement can only be annulled conditionally in this manner. The application program which logs out of the OpenCPIC manager under this name with *CMRLTP* cannot accept any more dialog requests for this local name until it is terminated (or until the local name is validated again with a corresponding *CMSLTP* call). However, the *locapro_name* in the configuration file remains valid. In other words, if the OpenCPIC manager receives another dialog request for this local name and no active AP of this name has logged in, the program generated with the LOCAPRO statement in the manner described under 3. is started again.

3.1.4 SYMDEST - Defining a communication partner

Syntax

SYMDEST	symdest_name,
PARTNER-APPL	**=** partappl_name**,**
PARTNER-APRO	**=** partner_name [**,**
PARTNER-TYPE	**=** partner_type][**,**
SEC-TYPE	**=** security_type][**,**
SEC-UID	**=** security_userid][**,**
SEC-PASS	**=** security_passwd]

Description

This statement is used to define a communication partner for a partner application. Such a partner can, for example, consist of a CPI-C application program if the carrier system of the partner is OpenCPIC, or a UTM-transaction code if the partner is a UTM application.

Every partner is described in a separate entry in the CPI-C side information. In CPI-C, such entries are referenced using a symbolic destination name.

symdest_name
: This is the symbolic destination name for the partner. It references the entry in the side information and is specified as an abbreviation for addressing the partner in the CPI-C call *Initialize_Conversation (CMINIT)* (parameter: *symdest_name*).

  The *symdest_name* consists of up to 8 characters from CPI-C character set 01134 (refer to the “section “Character sets” on page 191”), i.e. only upper-case letters and numbers must be used. Any *symdest_name* shorter than 8 characters is filled with blanks.

  ".DEFAULT" is also permissible as a *symdest_name* (refer to note 4).

partappl_name
: This is the name of the partner application via which the partner program is addressed. The *partappl_name* must be defined in a PARTAPPL statement in the generation file.

partner_name

This is the symbolic name of the partner program defined as local name on the partner side. During dialog establishment, the *partner_name* is specified as a parameter in the OSI-TP protocol element *TP-BEGIN-DIALOGUE-request*.

In OSI-TP terminology, this symbolic name corresponds to the TPSU title.

The *partner_name* can be up to 64 characters long. The number of permissible characters depends on the value of the parameter PARTNER-TYPE = *partner_type* (see below).

Note: If the partner is a UTM application, the *partner_name* must not be longer than 8 characters, and must only consist of upper-case letters and numbers.

partner_type

This indicates the ASN.1 data type (refer to ISO 8824 [29]) with which the *partner_name* was formed. This type is specified on coding of the *partner_name*. The following keywords are permissible:

= **PRINTABLE-STRING**

ASN.1 character-string-type *PrintableString*
The permissible characters for this data type are listed in the section “Character sets” on page 191.

= **T61-STRING**

ASN.1 character-string-type *T61String* (default value)
The permissible characters for this data type are listed in the section “Character sets” on page 191. (default value)

= **INTEGER** ASN.1-Type *Integer*

security_type

This indicates the access control information (security data) sent to the partner application during active dialog establishment (also refer to note 3 on page 46).

The following keywords can be specified:

= **NONE** No access control information is sent (default value).

= **SAME** The user ID under which the local application program was started is sent. A password is not entered.

= **PROGRAM** The user ID generated with *SEC UID = security_userid* and the password generated with *SEC PASS = security_passwd* are sent.

The parameter SEC TYPE = *security_type* is only evaluated if the application context name **utm-secu** was specified in the corresponding PARTAPPL statement (refer to the parameter APPLICATION-CONTEXT = *context_name*, page 35). In the case of every other application context, SEC-TYPE is set by default to NONE.

security_userid

This is the user ID sent as part of the access control information (see above, parameter SEC-TYPE). The *security_userid* can be up to eight characters long. It is part of the CPI-C side information and can be modified with the call *Set_Conversation_Security_User_ID (CMSCSU)*.

This parameter is only evaluated for SEC-TYPE = PROGRAM.

security_passwd

This is the password sent as part of the access control information (see above, parameter SEC-TYPE). The password can be up to eight characters long. It is part of the CPI-C side information and can be modified with the call *Set_Conversation_Security_Password (CMSCSP)*.

This parameter is only evaluated for SEC-TYPE = PROGRAM.

Notes

1. The data generated with the SYMDEST statement forms an entry in the CPI-C side information; the fields *AP_title*, *AE_qualifier* and *application_context_name* from the corresponding PARTAPPL statement (refer to the parameter PARTNER-APPL) are allocated here.

   The field *mode_name* in the side information is not supported (also refer to the section "The CPI-C interface of OpenCPIC" on page 77").

   Instead of a symbolic destination name generated with the SYMDEST statement, an empty symbolic destination name (consisting of 8 blank characters) can also be specified as the parameter *sym_dest_name* in the CPI-C call *Initialize_Conversation (CMINIT)*. In this case, the local CPI-C application program is responsible for supplying the required information. The partner application and the name of the partner application program must be declared in particular. The partner application is specified with one or more of the CPI-C calls *Set_AP_Title (CMSAPT)*, *Set_AE_Qualifier (CMSAEQ)* and *Set_Application_Context_Name (CMSACN)*. The name of the partner application program is set with *Set_TP_Name (CMSPTN)* (also refer to note 4).

   Even if a symbolic destination name generated with a SYMDEST statement is specified in *Initialize_Conversation (CMINIT)*, the entry in the side information of the conversation can be changed subsequently with *CMSAPT*, *CMSAEQ* or *CMSACN*. However, note that a modification to the side information using at least one of the above-mentioned CPI-C

calls invalidates the parameter PARTAPPL in the SYMDEST statement and, thus, all address information on the partner application available via this parameter (i.e. *AP_title*, *AE_qualifier* and *application_context_name)*. This means that if you change *AE_Qualifier* with *CMSAEQ*, for example, then the values of *AP_title* and *application_context_name* in the side information are deleted and might need to be set again.

In case of an *Allocate (CMALLC)* call, the OpenCPIC manager checks whether the partner application addressed via the current side information entry using *AP_title*, *AE_qualifier* and *application_context_name* is valid. If a valid, non-empty *symdest_name* was specified beforehand in *CMINIT* and the side information was not modified with *CMSAPT*, *CMSAEQ* or *CMSACN*, then the validity of the partner application is verified by the previously performed check of the configuration data. If however, the side information was modified by at least one of the three afore-mentioned calls between *CMINIT* and *CMALLC*, the current side information entry is used to check whether a matching partner application can be found in the configuration file. At least one of the three mentioned fields must be allocated and the combination of the allocated fields must uniquely identify a partner application generated with a PARTAPPL statement. If this is not the case, then the *CMALLC* is rejected with the return value *CM_PARAMETER_ERROR*. However, if the partner application is uniquely identifiable, then the entire address information from the localized PARTAPPL statement is used for dialog establishment.

The CPI-C calls *Extract_AP_Title (CMEAPT), Extract_AE_Qualifier (CMEAEQ)* and *Extract_Application_Context_Name (CMEACN)* can be used to extract the presently valid values from the side information.

Another possibility of addressing a partner application is provided by the CPI-C call *Set_Partner_LU_Name (CMSPLN)*. In OpenCPIC, this call can be used to specify the symbolic name of a partner application with which a link is to established using a subsequent *Allocate (CMALLC)*. If a corresponding PARTAPPL statement exists in the configuration, then the complete address information (i.e. *AP_title*, *AE_qualifier* and *application_context_name*) is copied from this statement to the side information entry.

The CPI-C call *Extract_Partner_LU_Name (CMEPLN)* can be used to extract the presently valid name of the partner application, provided that none of the parameters *AP_title*, *AE_qualifier* and *application_context_name* was changed.

Note: Although this type of addressing is more convenient than using *CMSAPT*, *CMSAEQ* and *CMSACN* described above, it is not portable. Officially, the CPI-C calls *CMSPLN* and *CMEPLN* are only supported for CRMs based on the LU6.2 protocol but not for OSI-TP CRMs. The support of these calls by OpenCPIC would require the X/Open CPI-C interface to be extended, and thus constitutes an exception.

2. During use of the calls *Set_Partner_LU_Name (CMSPLN)* and *Extract_Partner_LU_Name (CMEPLN)* the name of the partner application must not be longer than 17 characters, due to the definition of the CPI-C interface. Otherwise, *CMSPLN* returns the value *CM_PROGRAM_PARAMETER_CHECK*, while *CMEPLN* truncates the name after 17 characters.
3. The OSI-TP protocol does not support conversation security for CPI-C. For this reason, the calls *Set_Conversation_Security_Type (CMSCST)*, *Set_Conversation_Security_User_ID (CMSCSU)*, *Set_Conversation_Security_Password (CMSCSP)* and *Extract_Security_User_ID (CMESUI)* in the official CPI-C interface are only described for LU6.2 CRM. For deviations from this standard, OpenCPIC implements an internal protocol for exchanging access control information between OpenCPIC and UTM applications, and offers the mentioned CPI-C calls as an interface for managing the security data. However, their usage is only feasible with a UTM application as the communication partner. During dialog establishment, the access control information is sent as user data (initialization data) in *TP-BEGIN-DIALOGUE-request*. However, OpenCPIC does not evaluate any access control information received with a dialog request.
4. If the CPI-C program specified eight blank characters in the parameter *sym_dest.name* of the call *Initialize_Conversation (CMINIT)* and a *SYMDEST.DEFAULT* has been generated, then OpenCPIC copies the parameter generated for *.DEFAULT* to the conversation. This is an extension of X/Open CPI-C.

3.1.5 APPLICATION-CONTEXT - Defining an application context

Syntax

APPLICATION-CONTEXT	application_context_name**,**
OBJECT-IDENTIFIER	**=** context_objectid

Description

This statement is used to generate an application context. An application context specifies the number of rules to be agreed and observed by two communication partners when exchanging data and information via a dialog. The application context is agreed on during dialog setup and applies to all dialogs held via this association.

OpenCPIC requires an application context to contain at least the application service elements ACSE and TP-ASE. Other components of the application context are the abstract syntaxes and the transfer syntax in accordance with which the transmission data are coded and decoded.

An application context is identified by a unique object identifier which is assigned by an authorized registration agency (also refer to page 31).

The APPLICATION-CONTEXT statement is optional. Particularly if you only require connections to OpenCPIC or UTM partner applications, it is not necessary to define separate application context names. Instead, you can use one of the four, predefined application context names for communicating with the partner application (refer to the section “PARTAPPL - Defining the partner application” on page 34).

It is not possible to specify a name for a user-defined application context which is to contain abstract and transfer syntax of that application context. OpenCPIC always assigns user-defined application context names the abstract syntaxes UDT and - depending on the parameter CCR in the PARTAPPL statement - CCR, as well as the transfer syntax BER.

application_context_name

This is the symbolic name used in the generation file for referencing the application context. This name must be specified in the PARTAPPL statement (parameter APPL-CONTEXT) if you want to use your own application context instead of a predefined one.

The *application_context_name* can be up to eight characters long.

def-cont, **utm-secu** and **xatmi** are reserved for predefined application context names and must not be used.

context_objid
: This is the object identifier assigned to the application context (refer to page 31).

Notes

The application context name is a component of the side information; it can be modified with the CPI-C call *Set_Application_Context_Name (CMSACN)* and extracted with the call *Extract_Application_Context_Name (CMEACN)*. The object identifier is specified as the parameter or return value in each case here. More details on side information are provided in the description of in the SYMDEST statement, note 1.

3.2 Starting generation programs

The generation program *ocpic_gen* is used to create binary configuration files (format 1) from generation files and read generation data in ASCII form (format 2) out of configuration files. The terms generation file and configuration file are explained at the beginning of this chapter (page 29).

Syntax

ocpic_gen [gen_file] [**-f** conf_file] (Format 1)

ocpic_gen -r [conf_file] (Format 2)

Description

ocpic_gen evaluates the contents of the environment variable *OCPICDIR*. If *$OCPICDIR* has not been set, the path name of the OpenCPIC installation directory (shortly *openCPIC-path*) is used as the default value.

Format 1: Creating a binary configuration file

gen_file

This is the name of the generation file from which the generation data are to be read out. The generation file must be created using an editor. The structure of the generation file is described in section “Structure of the generation file” on page 30.
If *gen_file* is not specified, the generation data are read out of the standard file *$OCPICDIR/gen/genfile*.

conf_file

This is the name of the configuration file into which the generation data are to be written in binary form. If *conf_file* is not specified, the data is output to *$OCPICDIR/conffile*.

During its initialization, the OpenCPIC manager reads in the configuration data from *$OCPICDIR/conffile*. If you create the configuration file with *ocpic_gen -f* under a different name, you must rename it or copy it to *$OCPICDIR/conffile* before starting the OpenCPIC manager.

You can also create a new configuration file while the OpenCPIC manager is active. However, the new configuration data only become effective the next time the OpenCPIC manager is started.

Note for UTM users: As opposed to the UTM generation program *kdcdef*, *ocpic_gen* does not delete any recovery information from the *directory* *$OCPICDIR/SYNC* (refer to section “Recovery” on page 140. This information is always retained until the OpenCPIC manager is started next.

Format 2: Reading generation data out of a binary configuration file

conf_file

This is the name of the binary configuration file from which the generation data are to be read out. If *conf_file* is not specified, the data is read out of *$OCPICDIR/conffile*.

The generation data are written in legible form via *stdout* in the format described in section “Structure of the generation file” on page 30. You can route the output to a file and use it as a basis for a generation file.

The output of the generation data is complete, i.e. it also contains all optional parameters which might have been missing in the original generation file from which the configuration file was created.

Calling *ocpic_gen -r* can prove practical for testing a generation process.

3.3 A sample generation for OpenCPIC

On the installation of OpenCPI, the file *genfile* containing a sample generation is stored in the directory *openCPIC-path/gen*. In this sample generation, two partner applications are defined for a local OpenCPIC manager; one of these applications is also an OpenCPIC manager, while the other partner application consists of a UTM application. The internal security protocol is to be supported for links to the UTM partner.

The two application programs generated with the LOCAPRO statement can be addressed by programs of the partner application under this name and are then started automatically by the local application.

The three generated symbolic destination names allow simple addressing of the corresponding application programs of the communication partner.

Contents of the file genfile

```
* ------------------------------------------------------------------------------------
* OpenCPIC
* Sample generation file genfile
* ------------------------------------------------------------------------------------------------------------------------------------------------------


***** Local application: local OpenCPIC manager


LOCAPPL ocp_loc,
    APT = (2, 89, 111, 9),
    AEQ = 21


***** Partner application: remote OpenCPIC manager
***** Use default application context (1, 0, 10026, 6, 2) with CCR syntax included


PARTAPPL ocp_rem,
    APT = (2, 15, 99),
    AEQ = 100,
    APPL-CONTEXT = def-cont,
    ASSOCIATIONS = 10,
    CONTWIN = (2 4),
    CONNECT = 2,
    IDLETIME = 0,
    CCR = YES


***** Partner application: remote UTM application
***** CCR syntax always included
PARTAPPL utm_rem,
```

```
    APT = (2, 88, 235, 10),
    AEQ = 450,
    APPL-CONTEXT = utm-secu,
    ASSOCIATIONS = 10,
    CONTWIN = (2 4),
    CONNECT = 0,
    IDLETIME = 0


***** Local application programs: to be started automatically on receipt of
***** a conversation startup request from partner application

LOCAPRO sv_app_a,

    PATH = /home/ocpappl/server_a,
    LOGIN = guest

LOCAPRO sv_app_b,

    PATH = /home/ocpappl/server_b,
    LOGIN = guest


***** Symbolic destination names for active conversation startup by local
***** application programs


SYMDEST SYMDOCP0,

    PARTNER-APPL = ocp_rem,
    PARTNER-APRO = sv_app_0


SYMDEST SYMDOCP1,

    PARTNER-APPL = ocp_rem,
    PARTNER-APRO = sv_app_1


SYMDEST SYMDUTM0,

    PARTNER-APPL = utm_rem,
    PARTNER-APRO = UTMSVTAC,
    SEC TYPE = PROGRAM,
    SEC UID = OCPCLIEN,
    SEC PASS = OCPPASSW


* --- End of sample generation file genfile ----------------------------------------
```

4 Administering OpenCPIC

This chapter describes:

- How to start and terminate the OpenCPIC manager
- How to display status information via OpenCPIC
- How to manage associations for special applications
- How to record and edit trace information

$OCPICDIR working directory

OpenCPIC uses a working directory to store all the files and subdirectories required and generated during operation. The directory *openCPIC-path* is the default working directory. If a different working directory is to be used (see notice), you must set the environment variable *OCPICDIR* to the required (fully qualified) path name. The use of *OCPICDIR* is particularly necessary if you want to run several OpenCPIC managers simultaneously on one system. Every OpenCPIC manager requires its own separate working directory.

In the following, *$OCPICDIR* always implies the working directory of the OpenCPIC manager which you want to administer (i.e. *openCPIC-path*, if *OCPICDIR* is not set).

It is strongly recommended to create always a individual working directory, even if only one OpenCPIC manager is running on the system. Please note the following:

- The user ID used for the operation of the OpenCPIC manager must have write permission on *$OCPICDIR*.
- In addition, you must create two subdirectories with the names „PROT“ and „SYNC“ under *$OCPICDIR*.

4.1 Starting the OpenCPIC manager

Before starting, you must create the configuration file *conffile* and store it in the working directory *$OCPICDIR* as described in chapter 3.

The OpenCPIC manager is started using the shell procedure *ocpic_start*. Do not invoke the program *ocpic_mgr* directly, as this could lead to problems during initialization.

Syntax

ocpic_start [**-tp**] [**-tm**] [**-c**]

Description

ocpic_start starts the OpenCPIC manager as a background process.

-tp — This option is used to activate the XAP-TP trace on start of the OpenCPIC manager. As a result, trace information on the XAP-TP component is also recorded for diagnostic purposes. More details on this are provided in the "section "Recording trace information" on page 68".

-tm — This option is used to activate the manager trace on start of the OpenCPIC manager. As a result, trace information on the OpenCPIC manager is also recorded for diagnostic purposes. More details on this are provided in the section "Recording trace information" on page 68.

-c — This option is used to cold start the OpenCPIC manager. *ocpic_start* performs a warm start by default. This is necessary when you want to carry out a recovery after the occurrence of a system error. More details on the recovery process are provided in chapter "Global transactions" on page 137.

Note: When the OpenCPIC manager is cold started (switch *-c*) all the recovery information in the directory *$OCPICDIR/SYNC* is lost, except for the log damage records (refer to section "Recovery" on page 140). For this reason, you should ensure that the information in the log record is no longer required before you perform a cold start. If an error occurred in a previous transaction process and log records are present, you must carry out a warm start.

After a successful start, the following messages appear on the screen:

```
OpenCPIC manager: start
```

```
OpenCPIC [BD] 001 Start of the OpenCPIC manager with PID nnnn

Copyright (C)
2017 Fujitsu Technology Solutions GmbH
All rights reserved

OpenCPIC [BD] 016 OpenCPIC manager initialized
```

The OpenCPIC manager is now ready to receive requests from local application programs or remote applications. All other messages from the OpenCPIC manager are output to the file *$OCPICDIR/PROT/prot.mgr*.

4.2 Terminating the OpenCPIC manager

The OpenCPIC manager is terminated with the following command

ocpic_adm -e

The command *ocpic_adm* is a service program for administering OpenCPIC and is described fully on page 74. *ocpic_adm -e* sends the signal *SIGTERM* (15) to the OpenCPIC manager, which is then terminated correctly.

The same result is achieved with the following command:

kill -15 pid_mgr

The PID of the OpenCPIC manager is specified for *pid_mgr* here.

4.3 Displaying status information

The command *ocpic_sta* is used to display status information on the OpenCPIC manager.

Syntax

ocpic_sta [**-l**]

Description

ocpic_sta provides a standard output of the status information using the Unix/Linux command *pg*. You can scroll forward and backward through the displayed text with the corresponding *pg* commands.

If you invoke *ocpic_sta* without the switch, then brief information is output.

-l	This switch causes *ocpic_sta* to display detailed status information instead of brief information.

4.3.1 Output of ocpic_sta (brief information)

```
        OpenCPIC V4.0
    >> STATUS REPORT<<
```

```
PID of the OpenCPIC manager : nnnn
Local application : <name>

Partner application     Max asso        Established asso   Busy asso
partappl_name_1         <number>        <number>           <number>
...                     ...             ...                ...
partappl_name_n         <number>        <number>           <number>
```

Meaning of the output fields

PID of the OpenCPIC manager
: Process ID of the program *ocpic_mgr*

Local application
: Name of the local application, as specified with the LOCAPPL statement in the generation file.

Partner application
: Name of the partner application, as specified with the PARTAPPL statement in the generation file.

Max asso
: The maximum number of associations which can be set up for this partner application (parameter ASSOCIATIONS of the PARTAPPL statement in the generation file).

Established asso
: Number of presently established associations for this partner application.

Busy asso
: Number of presently busy associations for this partner application.

4.3.2 Output of ocpic_sta -l (detailed information)

```
OpenCPIC V4.0
>> STATUS REPORT <<
```

```
PID of the OpenCPIC manager : nnnn
Local application : <name>
```

Table of active application programs

AP-PID	Starter	TA status	Node position	Conversations
nnnn	<value>	<status>	<position>	<number>
...	...	...	...	...
nnnn	<value>	<status>	<position>	<number>

Table of partner applications and links

```
Partner application: <name>
Associations:                 Maximum = <number>
Contention winner:            Minimum = <number>,    Maximum = <number>
```

Association status	Content role	Dialog status	Functional units	TA mode	Conv. ID	Conv. status	AP-PID
<status>	<role>	<status>	<units>	<mode>	<id>	<status>	nnnn
...	...	...	...	...	...	...	...
<status>	<role>	<status>	<units>	<mode>	<id>	<status>	nnnn

```
Partner application: <name>
Ássociations:                 Maximum = <number>
Contention winner:            Minimum = <number>,    Maximum = <number>
...
```

Table of local TPSU titles

```
AP-PID nnnn:                    <number>
                                ...

AP-PID nnnn:                    <number>
                                ...

...
```

Table of waiting conversation startup requests

```
Partner application                 Local TPSU title
<name>                              <title>
...                                 ...
<name>                              <title>
```

Meaning of the output fields

PID of the OpenCPIC manager
: Process ID of the program *ocpic_mgr*

Local application
: Name of the local application, as specified with the LOCAPPL statement in the generation file.

Table of active application programs

AP-PID
: Process ID of the application program

Starter
: This specifies the initiator of the application program

`= MGR`
: The AP was started following the reception of a dialog request from the OpenCPIC manager.

`= USER`
: The AP was started with an operator command.

TA status
: This indicates the current status of the transaction node if the AP is involved in a transaction. If the AP is not involved in a transaction, this field remains empty.

  If the transaction node is approaching the precommit phase or already in it, and no *TP-COMMIT-request/TP_ROLLBACK-request* has been sent yet, or no *TP-COMMIT-indication/TP_ROLLBACK-*

indication has been received yet, then this value is always *IDLE*. The possible output values and their meanings are shown in the table of dialog statuses in the section titled "Status values of dialogs and transactions" on page 64.

Node position — This indicates the position of the AP in the transaction tree if the AP is presently involved in a transaction.

- = ROOT — The AP is the root node of the transaction
- = INTERMEDIATE — The AP is an intermediate node of the transaction
- = LEAF — The AP is a leaf node of the transaction

Conversations — This indicates the number of currently active conversations of the AP.

Table of partner applications and links

Partner application — Name of the partner application, as specified in the PARTAPPL statement in the generation file.

Associations — - Maximum
The maximum number of associations which can be set up for this partner application.

Contention winner — - Maximum
The maximum number of contention-winner associations which can be set up for this partner application.

- Minimum
The minimum number of contention-winner associations which can be set up for this partner application.

Association status — The current status of the association

- = aassrq — The association is in the setup phase. The local XAP-TP component sets up the association. A confirmation of successful setup is awaited from the partner application. This status is only intended to be temporary.

= bidding — The local XAP-TP component wants to open a dialog on a currently free association, for which this component is the contention loser. For this, it first sends a *TP-BID-RI* and then waits for a response from the partner application (i.e. for a *TP-BID-RC*). This status is only intended to be temporary.

= busy — The association has been set up and is reserved by a dialog.

= free — The association has been set up and is presently not reserved by any dialog.

= rsp stop — The association is in the setup phase. The remote XAP-TP component sets up the association. The local XAP-TP component waits for a *DATA-GO* signal from the OSS. This status is only intended to be temporary.

= waitgo — The association is in the setup phase. The local XAP-TP component waits for a *DATA-GO* signal from the OSS. This status is only intended to be temporary.

Content role — This indicates whether the local application is a contention loser or contention winner for this association.

= LOSER — The local application is the contention loser.

= WINNER — The local application is the contention winner.

dialog status — This indicates the current status of the dialog reserving the association. If the association is not reserved by any dialog, this field remains empty.
The possible output values and their meanings are shown in the table of dialog statuses in the "section "Status values of dialogs and transactions" on page 64".

Functional units — This indicates which OSI-TP functional units are supported for the dialog reserving this association. The supported functional units are selected by the partner applications during dialog setup and are dependent on the *sync_level* and *transaction_control* of the CPI-C conversation to be held via this dialog.
If the association is not reserved by a dialog, this field remains empty.
The various functional units are output as upper-case abbreviations and joined with a '+'. The abbreviations have the following meanings:

= D — *Dialog* functional unit
This functional unit is always supported.

= P — *Polarized Control* functional unit
This functional unit is always supported.

= H — *Handshake* functional unit
This functional unit is supported for *sync_level = CM_CONFIRM* and *sync_level = CM_SYNC_POINT*.

= C — *Commit* and *Chained Transactions* functional units
These functional units are supported for *sync_level = CM_SYNC_POINT, sync_level = CM_SYNC_POINT_NO_CONFIRM* and *transaction_control = CM_CHAINED_TRANSACTIONS*.

= U — *Commit* and *Unchained Transactions* functional units
These functional units are supported for *sync_level = CM_SYNC_POINT* ,. *sync_level = CM_SYNC_POINT_NO_CONFIRM* and *transaction_control = CM_UNCHAINED_TRANSACTIONS*.

The OSI-TP functional unit *Shared Control* is not supported in OpenCPIC.
The individual OSI-TP functional units) are described in ISO 10026 [32].

TA mode — This indicates whether the dialog reserving the association is currently involved in a transaction. If the association is not reserved by any dialog, this field remains empty.

= TRUE — The dialog is part of a transaction.

= FALSE — The dialog is not part of a transaction

Conv. ID — This indicates the ID of the conversation being held via the dialog reserving the association. If the association is not reserved by any dialog or no conversation is assigned to the dialog, this field remains empty.

Conv. status — This indicates the *conversation_state* of the conversation being held via the dialog reserving the association. If the association is not reserved by any dialog or no conversation is assigned to the dialog, this field remains empty.
The status values here correspond to those defined in the CPI-C specification [24] as follows:

= INITIALIZE	*CM_INITIALIZE_STATE*	(2)
= SEND	*CM_SEND_STATE*	(3)
= RECEIVE	*CM_RECEIVE_STATE*	(4)
= CONFIRM	*CM_CONFIRM_STATE*	(6)
= CONF_SEND	*CM_CONFIRM_SEND_STATE*	(7)
= CONF_DEAL	*CM_CONFIRM_DEALLOCATE_STATE*	(8)
= DEFER_RECV	*CM_DEFER_RECEIVE_STATE*	(9)
= DEFER_DEAL	*CM_DEFER_DEALLOCATE_STATE*	(10)
= SYNC_POINT	*CM_SYNC_POINT_STATE*	(11)
= SY_PO_SEND	*CM_SYNC_POINT_SEND_STATE*	(12)
= SY_PO_DEAL	*CM_SYNC_POINT_DEALLOCATE_STATE*	(13)
= INIT_INCOM	*CM_INITIALIZE_INCOMING_STATE*	(14)
= PREPARED	*CM_PREPARED_STATE*	(18)

AP-PID — This indicates the process ID of the application program holding a conversation via the dialog reserving the association. If the association is not reserved by any dialog, this field remains empty.

Table of the local TPSU title

AP-PID — This indicates the process ID of the local application program. Next to it is a list of all local names under which this AP has logged in, either with *Specify_Local_TP_Name (CMSLTP)* or by a LOCAPRO statement in the generation file on start of the OpenCPIC manager.

Table of waiting conversation startup requests

Partner application — Name of the partner application from which a dialog request was received.

Local TPSU title — Local name (corresponding to the TPSU title in OSI-TP or the *TP_Name* in CPI-C) for which the dialog request was received.

4.3.3 Status values of dialogs and transactions

In addition to the status of conversations, *ocpic_sta* indicates the status values of dialogs and transaction nodes.

The OpenCPIC manager maps CPI-C conversations to OSI-TP dialogs. For every conversation, an OSI-TP dialog is set up on which OSI-TP protocol elements are transmitted and received. The protocol elements and the status table for OSI-TP dialogs are defined in 10026-2 [32], Part 2.

In OpenCPIC, every dialog is managed by an XAP-TP-dialog entity. Another entity manages the transactions in which the local application programs are involved, and monitors the two-phase commit protocol for transaction completion. This entity is termed control entity.

The dialog state output by *ocpic_sta* corresponds to the actual value of the XAP-TP environment attribute *AP_STATE* of the related dialog entity. It indicates the status of the node in the dialog tree.
The transaction state output by *ocpic_sta* corresponds to the actual value of the XAP-TP environment attribute *AP_TP_STATE* of the control entity with respect to the related transaction. It indicates the status of the node in the transaction tree

XAP-TP [25] defines 35 different status values for dialog and control entities; statuses 1-25 correspond to the statuses for OSI-TP dialogs defined in ISO standard 10026-2.

The following table shows which status values are output by *ocpic_sta* and which OSI-TP statuses they correspond to. For statuses defined only in XAP-TP but not in ISO 10026, the corresponding column remains empty.

Dialog status	Description	ISO 10026
IDLE	No connection	1
WALLOCcnf	The LOCAL side tries to reserve an association.	
ALLOCATED	The LOCAL side has reserved an association and is attempting to establish a connection with a partner.	
DATA_XFER	The LOCAL side is authorized to transmit.	2
RECEIVE	The LOCAL side is not authorized to transmit.	3
ERROR_RECEIVE	The LOCAL side not authorized to transmit has sent at a *TP-U-ERROR-request* and is waiting for the transmission rights.	4
ERROR	The LOCAL side is authorized to transmit and has received a *TP-U-ERROR-indication*. The transmission rights are granted.	5

Dialog status	Description	ISO 10026
WHANDcnf	The LOCAL side has sent a *TP-HANDSHAKE-request* and is waiting for a *TP-HANDSHAKE-confirmation*.	6
WHANDrsp	The LOCAL side has received a *TP-HANDSHAKE-indication* but not yet sent a *TP-HANDSHAKE-response*.	7
WENDcnf	The LOCAL side has sent a *TP-END-DIALOGUE-request* with *confirmation=TRUE* and is waiting for a *TP-END-DIALOGUE-confirmation*.	11
WENDcnf	The LOCAL side has received a *TP-END-DIALOGUE-indication* with *confirmation=TRUE* but not yet sent a *TP-END-DIALOGUE-response*.	12
WHAND_GCcnf	The LOCAL side has sent a *TP-HANDSHAKE-AND-GRANT-CONTROL-request* and is waiting for a *TP-HANDSHAKE-AND-GRANT-CONTROL-confirmation*.	13
WHAND_GCrsp	The LOCAL side has received a *TP-HANDSHAKE-AND-GRANT-CONTROL-indication* but not yet sent a *TP-HANDSHAKE-AND-GRANT-CONTROL-response*.	14
WREADYind	The LOCAL side has sent a *TP-PREPARE-request* with *data_permitted=FALSE* and is waiting for a *TP-READY-indication*.	15
WREADY_DATAP	The LOCAL side has sent a *TP-PREPARE-request* with *data_permitted=TRUE* and is waiting for a *TP-READY-indication*.	16
READY	The LOCAL side has received a *TP-READY-indication*.	17
WPREP_ALLreq	The LOCAL side has received a *TP-PREPARE-indication* with *data_permitted=FALSE*.	18
WPREP_DATAP	The LOCAL side has received a *TP-PREPARE-indication* with *data_permitted=TRUE*.	19
PREPARING	The LOCAL side has sent a *TP-PREPARE-ALL-request*. This side has thus entered the first phase of the two-phase commit protocol (precommit phase).	
LOGGIN_READY	The LOCAL side has received a *TP-READY-ALL-indication* in the status *PREPARING*. The first phase of the two-phase commit protocol (precommit phase) is complete.	
WCOMMITind	The LOCAL side has sent a *TP-COMMIT-request* in the status *LOGGIN_READY*. This starts the second phase of the two-phase commit protocol (commit phase).	20

Dialog status	Description	ISO 10026
COM_WDONEreq	The LOCAL side has received a *TP-COMMIT-indication* in the status *WCOMMITind*. The local resource managers are requested to bring their data into the final state; this saves data modifications performed during the transaction. This status is also assumed when the local side receives a *TP-U-ABORT-indication* or *TP-P-ABORT-indication* in the status *WCOM_COMPind*.	21
WCOM_COMPind	The LOCAL side has sent a *TP-DONE-request* in the status *COM_WDONEreq*. All local resource managers have completed their data modifications.	22
ROL_WDONEreq	Before or during the precommit phase, the LOCAL side sent or received a protocol element which led to a rollback of the transaction.	23
WROL_COMPind	The LOCAL side has sent a *TP-DONE-request* in the status *ROL_WDONEreq*. All local resource managers have undone their data modifications.	24
ZOMBIE	Before commencement of the precommit phase, the LOCAL side received a protocol element which ended the dialog (e.g. *TP-U-ABORT-indication*); *Rollback=FALSE* was set here. Although the dialog no longer exists, the entity remains busy until the current transaction has been completed.	25
HEURISTIC_LO	This status is assumed by the transaction node following completion of the transaction, if inconsistencies between the involved transaction nodes arose as a result of heuristic decisions during the commit phase (refer to page 27).	

4.4 Clearing associations

As the system administrator, you can use the local application to clear existing associations to a partner application. For this, invoke the command *ocpic_adm* with switch *-d* (disconnect) or *-a* (abort).

Syntax

ocpic_adm {**-d** | **-a**} partappl_name

Description

-d	Use this switch to clear all existing free associations to the partner application *partappl_name*. An association is free when it is not reserved by any dialog.
-a	Use this switch to immediately terminate all existing associations to the partner application *partappl_name*, irrespective of whether or not they are busy in a dialog.
partappl_name	This is the symbolic name of the partner application, as generated with the PARTAPPL statement in the generation file.

Note

1. The current statuses of all existing associations can be viewed with the command *ocpic_sta -l* (refer to the section “Displaying status information” on page 56).
2. Associations no longer required can also be controlled via the generation parameter IDLETIME in the PARTAPPL statement. If you enter a value $t > 0$ here, an association is cleared automatically once it has remained free for longer than t seconds.

4.5 Recording trace information

Trace information serves as an aid for error diagnosis in OpenCPIC. It can be generated by activating a trace. The trace writes the information to special log files while the program is running.

The following traces are available in OpenCPIC:

1. XAP-TP trace
   When this trace is active, the entire OSI-TP protocol underneath the XAP-TP interface is recorded. The XAP-TP trace also contains the OSS trace.
2. Manager trace
   When this trace is active, internal runtime information on the OpenCPIC manager is logged above the XAP-TP interface.
3. CPI-C trace
   This trace is part of the library *libocpic.a* and creates a log of all CPI-C and TX calls in a CPI-C application program.
4. XATMI trace
   This trace is part of the XATMI library *libxtclt.a* and creates a log of all XATMI calls in an XATMI client program.

You can activate individual traces or several of them at the same time. Every trace creates its own log files. The log files of the XAP-TP and manager trace are created in the directory *$OCPICDIR/PROT*. The log files of the CPI-C and XATMI trace are created in the current working directory of the application program.

Notational conventions for log files

The individual traces create their log files under the following names:

XAP-TP trace:	**mgrlog.xap.**suff1.suff2
Manager trace:	**mgrlog.**suff
CPI-C trace:	**OCPIC**pid**.**suff
XATMI trace	**XTC**pid.suff

suff and *pid* are placeholders with the following meanings:

suff,suff1,suff2 integral file name suffix

The name of a log file is extended by an integral suffix which, beginning with 0, is incremented by 1 on every file change. A file change can be performed via an administration command, or automatically when a certain maximum threshold is exceeded (refer to the following section and the “section

"Activating CPI-C trace" on page 87). This results in a sequence of log files whose order is indicated by the suffix *suff*.

pid — Process ID of the application program

In the case of the log files of the CPI-C and XATMI trace, the process ID of the application program forms part of the file name to allow unique assignment between the log file and application program.

4.5.1 Activating the XAP-TP trace and manager trace

You can activate the XAP-TP trace and manager trace already when starting the OpenCPIC manager. This is done with the options *-tp* and *-tm* of the command *ocpic_start* (refer to the section "Starting the OpenCPIC manager" on page 54).

The XAP-TP trace and manager trace are controlled with the command *ocpic_adm -t* when the OpenCPIC manager is running:

Syntax

ocpic_adm -t {**ton** | **mon** | **bon** | **tof** | **mof** | **bof** | **tfl** | **mfl** | **bfl**}

Description

ton	XAP-TP trace on Activates the XAP-TP trace
mon	manager trace on Activates the manager trace
bon	both traces on Activates the XAP-TP trace and manager trace
tof	XAP-TP trace off Deactivates the XAP-TP trace
mof	manager trace off Deactivates the manager trace
bof	both traces off Deactivates the XAP-TP trace and manager trace
tfl	XAP-TP trace flush Changes the log file *mgrlog.xap.suff* to *mgrlog.xap.suff+1* in the case of XAP-TP trace.

mfl	manager trace flush Changes the log file *mgrlog.suff* to *mgrlog.suff+1* in the case of manager trace.
bfl	both traces flush Changes the log file *mgrlog.xap.suff* to *mgrlog.xap.suff+1* in the case of XAP-TP trace, and changes the log file *mgrlog.suff* to *mgrlog.suff+1* in the case of manager trace.

Notes

1. The deactivation and subsequent reactivation of a trace implicitly changes the corresponding log file.
2. The manager trace automatically changes the current log file once it has exceeded the maximum size of 512 KB.
   The XAP-TP component automatically changes the log file once a certain size has been reached. *suff2* is then incremented by 1. If the file is changed with an administration command (*ocpic_adm -t tfl* or *ocpic_adm -t bfl*), then *suff1* is incremented by 1.
3. By default, a maximum of ten log files are created for each trace.
   After the log file with suffix 9 (*mgrlog.9*) has been closed, the manager trace starts counting again at 0, and already existing log files are overwritten.
   In the case of XAP-TP trace, the suffix is incremented continuously, but the file *mgrlog.xap.suff-10* is deleted when the file *mgrlog.xap.suff* is opened.
   The maximum number of retained log files can be changed by setting the environment variable *OCP_TRACELIMIT* to the required value before the OpenCPIC manager is started. However, at least two log files are always created. If you set *OCP_TRACELIMIT* to 0, this means that there is no upper limit for the number of log files.

4.5.2 Deactivating CPI-C trace and XATMI trace

The traces of the libraries *libocpic.a* and *libxtclt.a* are controlled via environment variables.

The environment variables must be set and exported from the working environment of the application program before the program is started. For application programs which are started by the OpenCPIC manager, the environment variables should be set accordingly before the OpenCPIC manager is started.

The environment variables used for controlling the CPI-C trace are described in “section “Activating CPI-C trace” on page 87.

The environment variables used for controlling the XATMI trace are described in section “Activating XATMI trace” on page 116.

4.5.3 Editing trace information

The log files created by the traces contain binary information which first needs to be converted into a legible format. As the log files of different traces have different binary formats, various editing commands exist here.

Except for the interface trace (trace level 1), CPI-C trace and XATMI trace, the information recorded by the other traces serves mainly as diagnostic documentation for the service department. This information is of very low significance to the user. If you suspect an error in the operation of OpenCPIC, you should send the log files in **unedited form** to your local service department. For preliminary diagnosis however, it might be advisable for the user to separately edit and evaluate the trace information.

4.5.3.1 Editing the log files of the XAP-TP trace

The log files of the XAP-TP trace (*mgrlog.xap.suff*) are evaluated with the OSS program *openCPIC-path/oss/step*.

Syntax

step [**-h**] [**-d**] [**-l=**nnn[**k**]] [**-s=**{**n**|**l**|**m**|**h**}] [**-ps=**[**t**][**s**][**p**][**a**][**F**]] [**-cref=**n] [**-f=**hh[**:**mm[**:**ss]]] [**-t=**hh[**:**mm[**:**ss]]] tracefile [tracefile...]

Description

-h — All the switches of *step* are output, together with brief explanations. All other specified switches are ignored.

-d — This outputs all user data in hexadecimal format. By default, step tries to decode and edit all presentation, ACSE and FTAM syntaxes coded in accordance with the ASN.1 basic encoding rules (BER). This switch disables decoding of the user data.

-l=nnn[k] — *nnn* specifies the maximum length of the user data (in bytes or kilobytes) which can be edited. If this switch is omitted, then all user data is output in full length. The value range of *nnn* is 0 - 999. If nnn is followed by **k**, then the length is output in kilobytes, otherwise in bytes.

-s		Restriction of the output of information.
=n	(no)	All information is output.
=l	(low)	Passwords are not output.
=m	(medium)	User IDs, account numbers and passwords are not output (default value).
=h	(high)	Like ‘m’, except that file names are suppressed too.
-ps		This switch is used select the protocol layer whose information is to be output. The available flags can be combined in any required form.
t		All transport-system events; *T-DATAIN* events and all outgoing protocol elements remain invisible.
s		All session events
p		All presentation events
a		All ACSE events
F		All FTAM events
-cref=n		Only trace records containing the connection reference *n* are output.
-f		Only trace records created from the specified time onwards are output. The time is entered in the format hh:mm:ss (hours:minutes:seconds).
-t		Only trace records created up to the specified time are output. The time is entered in the format hh:mm:ss (hours:minutes:seconds).
tracefile		Name of the log file(s) to be evaluated.

4.5.3.2 Editing log files of the manager trace

The log files of the manager trace (*mgrlog.suff*) are edited with the following command:

Syntax

ocpic_adm -l tracefile

Description

tracefile	This is the Unix/Linux path name of the binary log file to be edited, e.g. *$OCPICDIR/PROT/mgrlog.0*

The edited data is output in standard form (*stdout*) and can be routed to a file.

4.5.3.3 Editing log files of the CPI-C trace and XATMI trace

How to edit the log files of the CPI-C trace is described in the section “Activating CPI-C trace” on page 87.

The log files of the XATMI trace need not be edited - they can be read directly.

4.6 Administration with ocpic_adm (overview)

This section provides a summarized description of the administration command *ocpic_adm*.

Syntax

ocpic_adm -t {**ton**|**mon**|**bon**|**tof**|**mof**|**bof**|**tfl**|**mfl**|**bfl**} | **-d** partappl_name | **-a** partappl_name | **-e** | **-l** tracefile

Description

-t	This controls the XAP-TP and manager traces (refer to the section “Recording trace information” on page 68)
-d	This clears all free associations to the partner application *partappl_name* (refer to the section “Clearing associations” on page 67)
-a	This clears all existing associations to the partner application *partappl_name* (refer to the section “Clearing associations” on page 67)
-e	This terminates the OpenCPIC manager (refer to the section “Terminating the OpenCPIC manager” on page 55)
-l	This edits the log file *tracefile* (refer to the section “Editing trace information” on page 71)

5 Creating CPI-C application programs

The implementation of CPI-C application programs for OpenCPIC requires you to be familiar with the interface CPI-C. Transaction-oriented operations also require a knowledge of the interface TX (transaction demarcation interface) and, in some cases , the interface for database access in your database system, e.g. SQL. The interface for database access is not part of OpenCPIC and not treated in this manual. Familiarity with the XA interface is beneficial here, although you need not invoke XA functions directly from your application program, as communications between the application program and the local resource manager via XA are carried out by the TX functions.

The details on CPI-C and TX stated in this chapter are based on the X/Open definition of the specified interfaces in each case. The corresponding X/Open documents can be found in the literature index at the end of this manual. CPI-C 2.0 is described in [7], TX is described in [26].

The CPI-C and TX calls are implemented in the library *libocpic.a*. Compared with the X/Open definitions, *libocpic.a* contains certain restrictions as well as extensions. The differences to the X/Open definition and the special aspects to be considered when programming with *libocpic.a* are described in the section “Implementing application programs with libocpic.a” on page 76.

To create an executable program, link the library *libocpic.a* with the object modules of your application program. If you want to perform the transactions using local resource managers, you also need to link the XA library of your database product. The XA calls for transaction management are issued internally by the TX functions. OpenCPIC offers a convenient method of creating application programs (refer to the section “Linking application programs” on page 85).

5.1 Implementing application programs with libocpic.a

This section provides general information on CPI-C and the TX interface of OpenCPIC, and describes the special aspects to be considered when implementing application programs with *libocpic.a*.

5.1.1 General information

Programmiing languages

OpenCPIC application program can be writtin in C or COBOL. The X/Open specifications CPI-C and TX contain special notes for C and COBOL programs.

Required include files

C programs require the file i *cpic.h* for the use of CPI-C calls. The file i *tx.h* is required for the use of TX calls.

COBOL programs require the file *CMCOBOL* (CPI-C definitions) as well as *TXINFDEF* and *TXSTATUS* (TX definitions).

For the XA connection the file *xa.h* is required. The source which is used for the XA connection is provided in the file *openCPIC-path/xa/xa_info.c*.

All the stated include files are stored in the directory *openCPIC-path/include* on the installation of OpenCPIC.

Use of signals

In OpenCPIC, the signal 13 (*SIGPIPE*) is used for controlling communications between the OpenCPIC manager and application program. Consequently, this signal must not be used or intercepted in OpenCPIC application programs. Apart from that, there are no restrictions on the use of signals.

Generating child processes

As the OpenCPIC manager uses the process IDs of local application programs to identify and manage them, CPI-C and TX calls in OpenCPIC application programs must not be forwarded to child processes. All CPI-C and TX calls must originate from the same process. In particular, automatically started application programs must issue their calls from the same process with which they were started by the OpenCPIC manager.

5.1.2 The CPI-C interface of OpenCPIC

OpenCPIC essentially allows an implementation of the interface CPI-C 2 defined by X/Open and described in [24]. This document should thus be referred to, particularly as regards the syntax and semantics of the CPI-C calls. The present section simply describes aspects deviating from the official X/Open documentation and the special features of the CPI-C interface of OpenCPIC.

CPI-C side information

As described in [24], the address data of every partner application can be described in separate entries in the CPI-C side information. Such entries in the side information are referenced via symbolic destination names which can be specified in the *Initialize_Conversation (CMINIT)* call.

In OpenCPIC, you can create side information entries using the generation statement SYMDEST. The symbolic name of the SYMDEST statement corresponds to the symbolic destination name in CPI-C.

The attributes of the CPI-C side information described in [24] correspond to the following generation parameters in OpenCPIC.:

Attribute	**Generation parameters in the SYMDEST statement**
AP_title	PARTNER-APPL (parameter APT in the corresponding PARTAPPL statement)
AE_qualifier	PARTNER-APPL (parameter AEQ in the corresponding PARTAPPL statement)
application_context_name	PARTNER-APPL (parameter APPL-CONTEXT in the corresponding PARTAPPL statement)
partner_LU_name	PARTNER-APPL
TP_name	PARTNER-APRO
mode_name	Not supported (default value: eight blanks)
conversation_security_type	SECTYPE
security_user_ID	SECUID
security_password	SECPASS

Transmitting and receiving user data

Using the calls *Send_Data (CMSEND)* and *Receive (CMRCV)* a maximum of 32767 bytes of user data can be transmitted and received respectively. This maximum value can also be extracted with *Extract_Maximum_Buffer_Size (CMEMBS)*.

User data and status information are always returned in separate *Receive (CMRCV)* calls. In other words, if the value of *data_received* returned by *CMRCV* is not equal to *CM_NO_DATA_RECEIVED*, then *status_received* is always equal to *CM_NO_STATUS_RECEIVED*. Consequently a conversation can never assume the state *CM_SEND_PENDING_STATE*.

Overview of CPI-C calls

The following table provides an overview of all calls defined in CPI-C 2.0. The column titled“OpenCPIC” indicates whether calls are supported in OpenCPIC.

A“+” in this column means that the call in question is supported and conforms precisely with the X/Open definition.

A“*” implies special features and/or restrictions compared with X/Open CPI-C 2.0. These are described individually by the subsequent remarks. The number in the column titled “Remarks” refers to the corresponding remark below the table.

A “-” in the column titled “OpenCPIC” means that the call is not supported in OpenCPIC and always returns the value *CM_CALL_NOT_SUPPORTED*, unless stated otherwise in the remarks.

Table of CPI-C 2.0 calls

CPI-C 2.0 call		OpenCPIC	Remarks
Accept_Conversation	(CMACCP)	*	(1)
Accept_Incoming	(CMACCI)	*	(1)
Allocate	(CMALLC)	+	
Cancel_Conversation	(CMCANC)	+	
Confirm	(CMCFM)	+	
Confirmed	(CMCFMD)	+	
Convert_Incoming	(CMCNVI)	*	(2)
Convert_Outgoing	(CMCNVO)	*	(2)
Deallocate	(CMDEAL)	+	
Deferred_Deallocate	(CMDFDE)	+	
Extract_AE_Qualifier	(CMEAEQ)	+	

CPI-C 2.0 call		OpenCPIC	Remarks
Extract_AP_Title	(CMEAPT)	+	
Extract_Application_Context_Name	(CMEACN)	+	
Extract_Conversation_State	(CMECS)	+	
Extract_Conversation_Type	(CMECT)	+	
Extract_Initialization_Data	(CMEID)	-	
Extract_Maximum_Buffer_Size	(CMEMBS)	+	
Extract_Mode_Name	(CMEMN)	*	(3)
Extract_Partner_LU_Name	(CMEPLN)	*	(4)
Extract_Secondary_Information	(CMESI)	-	
Extract_Security_User_ID	(CMESUI)	*	(5)
Extract_Send_Receive_Mode	(CMESRM)	-	
Extract_Sync_Level	(CMESL)	+	
Extract_TP_Name	(CMETPN)	+	
Extract_Transaction_Control	(CMETC)	+	
Flush	(CMFLUS)	+	
Include_Partner_In_Transaction	(CMINCL)	+	
Initialize_Conversation	(CMINIT)	+	
Initialize_For_Incoming	(CMINIC)	+	
Prepare	(CMPREP)	+	
Prepare_To_Receive	(CMPTR)	+	
Receive	(CMRCV)	+	
Receive_Expedited_Data	(CMRCVX)	-	
Release_Local_TP_Name	(CMRLTP)	+	
Request_To_Send	(CMRTS)	+	
Send_Data	(CMSEND)	+	
Send_Error	(CMSERR)	+	
Send_Expedited_Data	(CMSNDX)	-	
Set_AE_Qualifier	(CMSAEQ)	*	(6)(7)
Set_Allocate_Confirm	(CMSAC)	+	
Set_AP_Title	(CMAPT)	*	(6)(8)
Set_Application_Context_Name	(CMSACN)	*	(6)

CPI-C 2.0 call		OpenCPIC	Remarks
Set_Begin_Transaction	(CMSBT)	+	
Set_Confirmation_Urgency	(CMSCU)	+	
Set_Conversation_Security_Password	(CMSCP)	*	(5)
Set_Conversation_Security_Type	(CMSCST)	*	(5)
Set_Conversation_Security_User_ID	(CMSCSU)	*	(5)
Set_Conversation_Type	(CMSCT)	*	(9)
Set_Deallocate_Type	(CMSDT)	+	
Set_Error_Direction	(CMSED)	+	
Set_Fill	(CMSF)	+	
Set_Initialization_Data	(CMSID)	-	
Set_Join_Transaction	(CMSJT)	+	
Set_Log_Data	(CMSLD)	-	(10)
Set_Mode_Name	(CMSMN)	*	(3)
Set_Partner_LU_Name	(CMSPLN)	*	(4)
Set_Prepare_Data_Permitted	(CMSPDP)	+	
Set_Prepare_To_Receive_Type	(CMSPTR)	+	
Set_Processing_Mode	(CMSPM)	+	
Set_Queue_Callback_Function	(CMSQCF)	-	
Set_Queue_Processing_Mode	(CMSQCM)	-	
Set_Receive_Type	(CMSRT)	+	
Set_Return_Control	(CMSRT)	+	
Set_Send_Receive_Mode	(CMSSRM)	-	
Set_Send_Type	(CMSST)	+	
Set_Sync_Level	(CMSSL)	+	
Set_TP_Name	(CMSTPN)	+	
Set_Transaction_Control	(CMSTC)	+	
Specify_Local_TP_Name	(CMSLTP)	+	
Test_Request_To_Send_Received	(CMTRTS)	+	
Wait_For_Completion	(CMWCMP)	-	
Wait_For_Conversation	(CMWAIT)	+	

Remarks

1. **Accept_Conversation(CMACCP), Accept_Incoming (CMACCI）**

   In principle, CPI-Callows an application program to accept several conversations using *Accept_Conversation (CMACCP)* and *Accept_Incoming (CMACCI)* and thus simultaneously hold several conversations with superior partners. This “multiple accept” is also supported by OpenCPIC but with the following restriction: As an OpenCPIC application program can only be involved in one transaction at a time (i.e. all active protected conversations of an AP belong to the same global transaction) and every node in a transaction tree always has exactly one superior (except for the root node), an OpenCPIC application program cannot hold more than one protected conversation with a superior.
   Furthermore, a transaction tree can only be extended downwards, i.e. by adding subordinates. In other words, an AP involved in a transaction cannot accept conversations from a superior, even if it does not yet have a superior.

   Consequently, a *CMACCP* or *CMACCI* is always responded to with the return value *CM_PRODUCT_SPECIFIC_ERROR* in OpenCPIC if at least one of the following conditions is true:
   – The AP has at least one active conversation with the *sync_level* C*M_SYNC_POINT* or *CM_SYNC_POINT_NO_CONFIRM*.
   – The AP has already issued *tx_begin()* and is thus involved in a transaction.

   In the case of *CMACCI*, the state of the conversation does not change as a result, but retains the value *CM_INITIALIZE_STATE*.

2. **Convert_Incoming (CMCNVI), Convert_Outgoing (CMCNVO)**

   These calls perform a code conversion from EBCDIC to ASCII (*CMCNVI*) or from ASCII to EBCDIC (*CMCNVO*). The conversion is carried out using the tables from the IBM document titled“NetView/PC Application Programming, Version1.2” [33]. The character sets in these tables conform with the character sets in CPI-C 2.0. The conversion tables are contained in the appendix (page 194).

3. **Extract_Mode_Name (CMEMN), Set_Mode_Name (CMSMN）**

   OSI-TP and XAP-TP do not support any mode names. The comparable XAP-TP attribute “quality-of-service” (QOS) is not supported in the XAP-TP implementation underlying OpenCPI, as the OSS interface does not contain the QOS. Consequently, it is pointless to define mode names in OpenCPIC application programs. Although *CMSMN* accepts all syntactically correct mode names, this call does not have any effect. *CMEMN* always returns a standard mode name consisting of eight blanks.

4. **Extract_Partner_LU_Name (CMEPLN), Set_Partner_LU_Name (CMSPLN)**

   Normally, these calls are only of significance for an LU6.2 CRM. To address the partner application in OSI-TP, CPI-C 2.0 offers the calls *Set_AP_Title (CMSAPT)*, *Set_AE_Qualifier (CMSAEQ)* and *Set_Application_Context_Name (CMSACN)* or *Extract_AP_Title (CMEAPT)*, *Extract_AE_Qualifier (CMEAEQ)* and *Extract_Application_Context_Name (CMEACN)*. However, as the calls *Set_Partner_LU_Name (CMSPLN)* and *Extract_Partner_LU_Name (CMEPLN)* are used for symbolic addressing in the preceding version OpenCPIC V2.0, they are still supported by the current version of OpenCPIC for reasons of compatibility. In OpenCPIC, the *partner_LU_name* in the CPI-C side information corresponds to the symbolic name of the partner application as generated with the PARTAPPL statement (refer to page 34). Deviating from the CPI-C description, it has a maximum length of 17 characters.

   The method of symbolic addressing via *CMEPLN* and *CMSPLN* supported in OpenCPIC is incompatible with certain other applications based on OSI-TP. To ensure portability, it is therefore advisable to address the partner application with *CMSAPT*, *CMSAEQ* and *CMSACN* in newly developed application programs.

5. **Extract_Security_User_ID (CMESUI),**
   Set_Conversation_Security_Password (CMSCSP),
   Set_Conversation_Security_Type (CMSCST),
   Set_Conversation_Security_User_ID (CMCSU)

   The OSI-TP protocol does not support conversation security for CPI-C. For this reason, the calls *Set_Conversation_Security_Type (CMSCST)*, *Set_Conversation_Security_User_ID (CMSCSU)*, *Set_Conversation_Security_Password (CMSCSP)* and *Extract_Security_User_ID (CMESUI)* in the official CPI-C interface are only described for LU6.2 CRM. For deviations from this standard, OpenCPIC implements an internal protocol for exchanging access control information between OpenCPIC and UTM applications, and offers the mentioned CPI-C calls as an interface for managing the security data. However, their usage is only feasible with a UTM application as the communication partner. During generation of the partner application with the PARTAPPL statement, the parameter APPL-CONTEXT = utm-secu needs to be set.
   The access control information is sent as user data (initialization data) in the *TP-BEGIN-DIALOGUE-request* during dialog establishment. However, OpenCPIC does not evaluate any access control information received with a dialog request.

6. **Extract_AE_Qualifier (CMEAEQ),**
   Extract_AP_Title (CMEAPT),
   Extract_Application_Context_Name (CMEACN),
   Set_AE_Qualifier (CMSAEQ),
   Set_AP_Title (CMSAPT),
   Set_Application_Context_Name (CMSACN)

   In OSI-TP, a partner application is addressed uniquely with the application process title (APT) and the application entity qualifier (AEQ). In addition, an application context must be agreed for every partner application. For each of these addressing parameters, X/Open CPI-C defines a field in the CPI-C side information (refer to the page 77). This allows *AP_title*, *AE_qualifier* and *application_context_name* to be specified by defining a symbolic destination name *(*SYMDEST statement*)* which is transferred as the parameter *symdest_name* with *CMINIT*.
   The functions *Set_AE_Qualifier (CMSAEQ)*, *Set_AP_Title (CMSAPT)* and *Set_Application_Context_Name (CMSACN)* can be used to change entries in the side information and address partner applications without using a symbolic destination name (transfer of an empty *symdest_name* to *CMINIT*). However, note that in OpenCPIC, every partner application must be defined by a PARTAPPL statement in the generation file. In other words, the address information transferred via *CMSAEQ*, *CMSAPT* and *CMSACN* must uniquely identify a partner application in the generation file .
   How the OpenCPIC manager checks address information is described in the section titled “SYMDEST - Defining a communication partner”, note 1 on page 44.

   The CPI-C calls *Extract_AP_title (CMEAPT)*, *Extract_AE_Qualifier (CMEAEQ)* and *Extract_Application_Context_Name (CMEACN)* can be used to extract the presently valid values from the side information.

7. **Set_AE_Qualifier (CMSAEQ)**

   OpenCPIC firmly specifies the value *CM_INT_DIGITS* for the parameter *AE_qualifier_format*. The parameter *AE_qualifier* must consist of an integer between 1 and 67 108 863. If an invalid value or the format *CM_DN* is transferred, the call returns the value *CM_PRODUCT_SPECIFIC_ERROR* .

8. **Set_AP_Title (CMSAPT)**

   OpenCPIC firmly specifies the value *CM_OID* for the parameter *AP_title_format*. The parameter *AP_title* must consist of an object identifier. If an invalid value or the format *CM_DN* is transferred, the call returns the value *CM_PRODUCT_SPECIFIC_ERROR* .

9. **Set_Conversation_Type (CMSCT)**

   OpenCPIC only allows the default value *CM_MAPPED_CONVERSATION* as a parameter for *conversation_type*.
   Calls with *conversation_type = CM_BASIC_CONVERSATION* are responded to with the return value *CM_PRODUCT_SPECIFIC_ERROR*.

10. **Set_Log_Data (CMSLD)**

    OpenCPIC does not support any log data. Consequently, this call always returns the value *CM_PRODUCT_SPECIFIC_ERROR*.

5.1.3 The TX-interface of OpenCPIC

OpenCPIC implements all functions defined in the X/Open interface TX. For more details, refer to the X/Open documentation [26].

Table of TX calls (C)

```
int tx_begin(void)
int tx_close(void)
int tx_commit
int tx_info(TXINFO *info)
int tx_open(void)
int tx_rollback(void)
int tx_set_commit_return(COMMIT_RETURN when_return)
int tx_set_transaction_control(TRANSACTION_CONTROL control)
int tx_set_transaction_timeout(TRANSACTION_TIMEOUT timeout)
```

Table of TX calls (COBOL)

```
CALL "TXBEGIN" USING TX-RETURN-STATUS
CALL "TXCLOSE" USING TX-RETURN-STATUS
CALL "TXCOMMIT" USING TX-RETURN-STATUS
CALL "TXINFORM" USING TX-INFO-AREA TX-RETURN-STATUS
CALL "TXOPEN" USING TX-RETURN-STATUS
CALL "TXROLLBACK" USING TX-RETURN-STATUS
CALL "TXSETCOMMITRET" USING TX-INFO-AREA TX-RETURN-STATUS
CALL "TXSETTIMEOUT" USING TX-INFO-AREA TX-RETURN-STATUS
CALL "TXSETTRANCTL" USING TX-INFO-AREA TX-RETURN-STATUS
```

5.2 Linking application programs

To create an executable application program, you need to link the object modules of your application program with the library *libocpic.a*.

You can create OpenCPIC programs in the programming language C/C++ or COBOL.

An include or COBOL COPY is supplied for program support for the programming languages C/C++ and COBOL, see “Creating Applications with X/Open Interfaces” in the openUTM manual.

Proceed as follows:

1. Compile your OpenCPIC programs with the C/C++ or COBOL compiler, specifying the necessary compiler options.
2. Link your executable OpenCPIC application program with the following components:
   - Objects of your OpenCPIC programs
   - OpenCPIC library under *openCPIC-path/libocpic.a*
   - OpenCPIC XA library under *openCPIC-path/xa/ocpic_xa_connect*

     In this way you control whether or not the application program can access a local resource manager. For details, see section “Local resource managers - XA connection” on page 137.
   - Further libraries required

If several resource managers are installed on your system, you must positively ensure that the OpenCPIC manager uses the same resource manager as the local application programs do. In other words, the program file *ocpic_mgr* and the local application programs must use the same XA connection and must reference the same resource manager.

5.3 Starting application programs

Before an OpenCPIC application program is started, the OpenCPIC manager must be activated.

Application programs can either be started manually with an operator command or automatically by the OpenCPIC manager on the reception of a dialog request.

Automatic starting only proves practical for application programs which accept dialog requests from the partner program with *Accept_Conversation (CMACCP)* and *Accept_Incoming (CMACCI)*. These are generally application programs which act as servers in a client-server system and only need to be started when client requests are issued.

All application programs to be started by the OpenCPIC manager must be defined in the generation file with a LOCAPROstatement (refer to page 39).

5.3.1 Environment variables

Before an OpenCPIC application program is started, it might be necessary to set certain required environment variables. Automatically started programs copy these environment variables from the working environment of the OpenCPIC manager. In this case, all the required environment variables must be set before the OpenCPIC manager is started.

– The named pipes required for communications between the application program and the OpenCPIC manager are created in the directory *$OCPICDIR*. The contents of the variable *OCPICDIR* in the environment of the application program must therefore be identical to those of *OCPICDIR* for the OpenCPIC manager. If *OCPICDIR* is not set, both the OpenCPIC manager as well as the library *libocpic.a* use the default value *openCPIC-path*.

– The environment variables *CPICTRACE*, *CPICPATH*, *CPICSIZE*, *OCP_TRACELIMIT* are used to control the CPI-C trace (refer to the section “Activating CPI-C trace” on page 88)

– All the environment variables required by the XA library of your resource manager must also be set.

Working environment of automatically started application programs

The OpenCPIC manager starts an application program with the system call *fork()*. The application program receives the following working environment:

– The user ID and group are determined with the LOGIN parameter of the LOCAPRO statement in the generation file.

– The working directory of the application program is the *HOME* directory of the user ID (*LOGIN* parameter).

– All file descriptors (including *stdin*, *stdout* and *stderr*) are closed during process generation by *fork()*.

– The environment variables from the working environment of the OpenCPIC manager (e.g *OCPICDIR*, *OCP_TRACELIMIT*, *CPICTRACE*) are copied. The environment variables *USER*, *LOGNAME*, *HOME* and *PATH* are set in accordance with the LOGIN parameter of the LOCAPRO statement.

5.3.2 Activating CPI-C trace

For diagnostic purposes, you can activate the trace of the library *libocpic.a* (CPI-C trace). As a result, trace information is recorded by *libocpic.a* while the program is running. The CPI-C trace operates at two trace levels: interface trace (trace level 1) and internal trace (trace level 2). For error diagnosis in application programs, the interface trace is generally sufficient.

The protocol files of the CPI-C trace are stored in the current working directory or in the directory specified with the environment variable *CPICTRACE* (refer to the following section “Controlling CPI-C trace” on page 88). The name of a log file consists of the following elements:

OCPICpid.suff

suff and *pid* are placeholders with the following meaning:

suff — integral file name suffix

The name of a log file is extended by an integral suffix which, beginning with 0, is incremented by 1 on every file change. A file change takes place automatically when a certain maximum threshold is exceeded; the value of this threshold is set with the environment variable *CPICTRACE* (refer to the following section “Controlling CPI-C trace” on page 88). This results in a sequence of log files whose order is indicated by the suffix *suff*.

pid — Process ID of the application program

Controlling CPI-C trace

The CPI-C trace is controlled via environment variables. The following environment variables are evaluated by the library *libocpic.a*:

CPICTRACE The environment variables must be set to activate the CPI-C trace. Two trace levels are available here:

=-[**s**|**S**] This activates the interface trace (trace level 1). This trace writes all CPI-C and TX calls with their parameters and return values into the log file. It is particularly suitable for error diagnosis in application programs.

=-[i| I] This activates the internal error trace (trace level 2). This trace contains the interface trace and also writes numerous items of internal information from the library into the log file. It should only be used if an error in the operation of OpenCPIC is suspected.

CPICPATH=pathname
This specifies the directory in which the log files of the CPI-C trace are to be stored. If *CPICPATH* is not set, the log files are stored in the current working directory of the application program.
Note : If *pathname* is not a valid directory on your system or the log files cannot be created in the specified directory, the trace remains inactive.

CPICSIZE=size This specifies the maximum size of the log files (in bytes). Once the log file *OCPIC<pid>.suff* has exceeded this size, a change is performed automatically to the file *OCPIC<pid>.suff+1*. If *CPICSIZE* is not set or *size* is not an integer, the default value of 524288 bytes (512 KB) is used.

OCP_TRACELIMIT=limit
This specifies the maximum number of log files to be created by CPI-C trace. *limit* is an integer between 0 and 1024.
By default, a maximum of ten log files are created for each trace. After the log file with suffix (*OCPIC<pid>.9*) has been closed, counting starts again at 0 and already existing log files are overwritten.
The maximum number of retained log files can be changed by setting the environment variable *OCP_TRACELIMIT* to the required value before the OpenCPIC manager is started. However, at least two log files are always created. If you set *OCP_TRACELIMIT* to 0, this means that there is no upper limit for the number of log files.

Editing log files

The log files of the CPI-C trace (*OCPIC<pid>.suff*) are evaluated with the following command:

Syntax

ocpic_tredit tracefile

Description

tracefile — This is the path name of the binary log file to be edited e.g. *$HOME/OCPIC1244.0*

The edited data is output in standard form (*stdout*) and can be routed to a file.

6 Creating XATMI client programs

XATMI (X/Open application transaction manager interface) is a standardized X/Open program interface for a communication resource manager which allows client-server communications with the (optional) use of transaction functions.

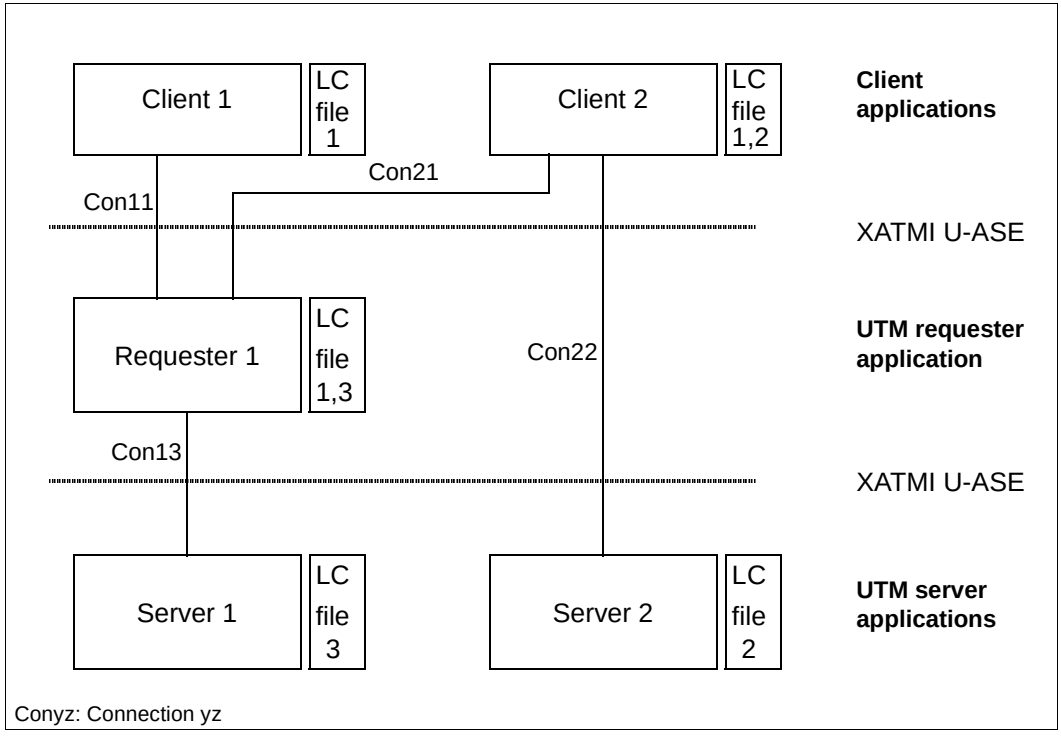
The XATMI library implemented in OpenCPIC is based on the CPI-C interface. In other words, XATMI programs which use XATMI calls must be linked with the XATMI library as well as the CPI-C library of OpenCPIC (*libocpic.a*).

The XATMI program interface for OpenCPIC is based on the X/Open XATMI specification [28]. This chapter describes the XATMI interface for clients using the OpenCPIC carrier system. You need to be familiar with this specification before starting this chapter, which uses the following essential terms:

Client	An application which calls service functions.
Server	A UTM application which contains service functions in C and/or COBOL. The service functions can consist of several subroutines.
Service	A service function programmed in C or COBOL in accordance with the XATMI specification. XATMI differentiates between two types of service: end services and intermediate services. An end service is only linked with its client and does not invoke any other services. An intermediate service invokes one or more other services.
Request	A request is an invocation of a service. It can be issued from a client or an intermediate service.
Requester	The XATMI specification uses the term "requester" for any application which invokes a service. The application can be a client or a server.
Typedbuffer	This is a buffer for exchanging typed and structured data between communications partners. These typed buffers implicitly declare the structure of the exchanged data to the carrier system and the application, and automatically match (code/decode) this structure in a heterogeneous network).

6.1 Client-server application network

The following diagram shows a client-server application network in which clients, servers and requesters communicate with each other. They exchange their typed data structures (typed buffers) in accordance with the protocol of the "XATMI U-ASE definition".



In any heterogeneous application network, the servers and clients require a local configuration defined in the respective local configuration file (LCF). The local configuration describes the services and their corresponding data structures, i.e.:

– For a server: all services which can be called up
– For a client: the services of all servers with which the client communicates
– For a requester (intermediate service): all available services as well as all used services

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

Several communications models are available for establishing client-server connections Con11, Con13,. .. .

6.1.1 Default server

To simplify client-server configurations, openUTM offers the possibility of specifying a default server using DEST=.DEFAULT in the SVCU statement of the local configuration file.

If a service *svcname2* which does not possess any SVCU entry in the local configuration file is used with the call `tpcall`, `tpacall` or `tpconnect`, the following entry is automatically used:

```
SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=.DEFAULT, MODE=RR
```

For this, OpenCPIC generation requires the following entry in the generation file:

```
SYMDEST .DEFAULT
PARTNER-APPL=...
PARTNER-APRO=...
```

6.2 Communications models

Three communications models are available to the programmer for client-server communications:

– Synchronous request response model: single-step dialog.
The client is blocked between the transmission of a service request and the arrival of the response.

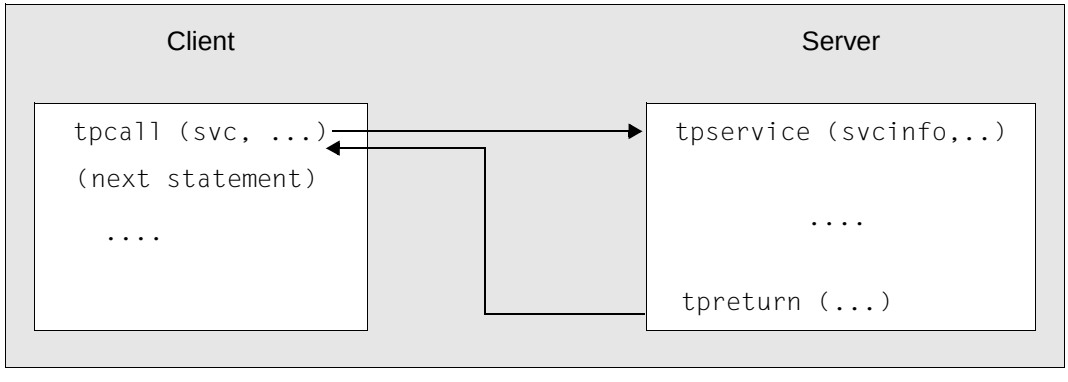
– Asynchronous request response model: single-step dialog.
The client is not blocked after the transmission of a service request.

– Conversational model: multi-step dialog.
The client and server can exchange data as required.

The XATMI functions required for these communication models are only outlined using C notation in the following. A detailed description of XATMI functions is provided in the XATMI specification [28].

6.2.1 Synchronous request response model

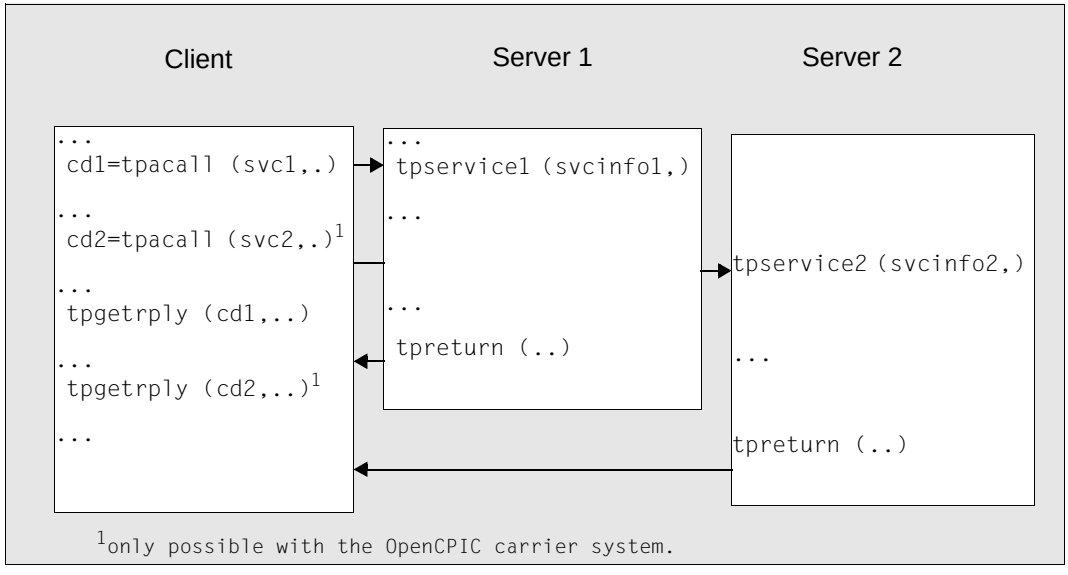
Such a service is called with a *tpcall* function. A *tpcall* addresses the service, sends exactly one message to it, and waits until it has received a response, i.e. *tpcall* has a blocking action.



svc identifies the internal name of the service, *svcinfo* the service info structure related to the service name, and *tpservice* the program name of the service routine. The template *tpservice()* is available for the programming language C in this context. For COBOL, the XATMI call TPSVCSTART needs to be used. The service info structure is defined in the XATMI interface and is supplied internally by XATMI.

6.2.2 Asynchronous request response model

In this model, communications take place in two steps. The first call (*tpacall*) addresses the service and sends the message, the second call (*tpgetrply*) is used later to fetch the response. In this model, several services can be addressed simultaneously. The following diagram elucidates the structure here:



svc1, *svc2* identify the internal names of the services, *cd1* and *cd2* the process-local communication descriptors, *svcinfo1* and *svcinfo2* the service info structures related to the service names, and *tpservice1, tpservice2* the program names of the service routines.

tpacall does not have a blocking action, i.e. the client can handle other local processes while waiting.

tpgetrply, in contrast, has a blocking action, i.e. the client waits until a response has been received.

In this model, a dialog TAC must be generated on the UTM server side for the service (as in the case of the synchronous request response model).

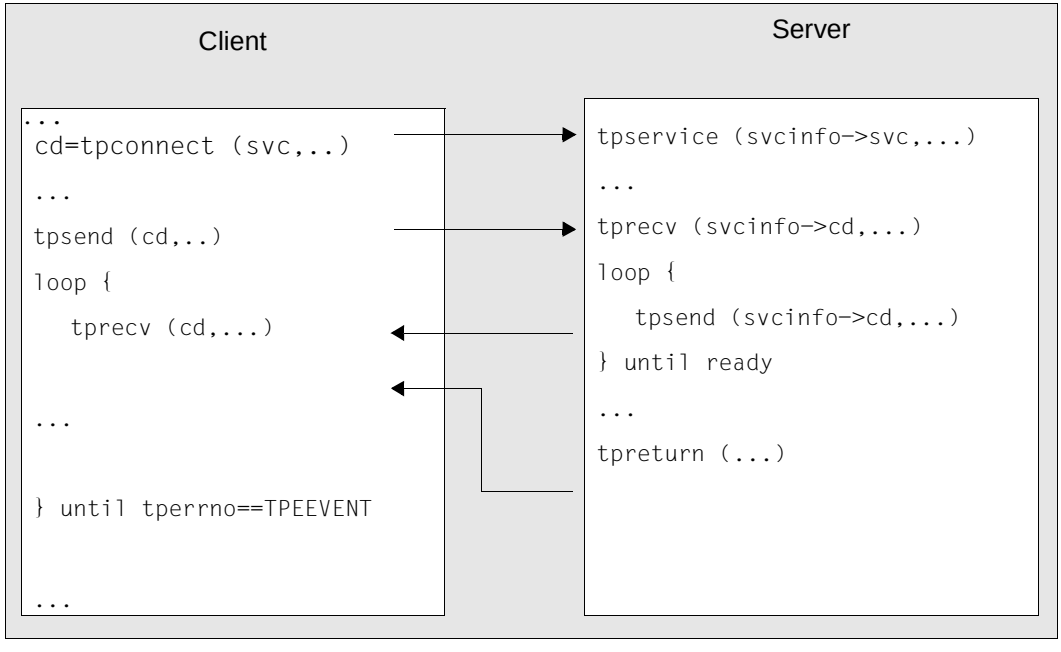
OpenCPIC allows several jobs to be processed simultaneously.

6.2.3 Conversational model

XATMI offers a conversational model for communications-oriented operations ("conversations").

This model can be used, for example, to transfer large amounts of data in several partial steps. If the synchronous *tpcall* is used here, problems might arise due to the restrictions for local data buffers.

Using the conversational model, a conversation with a service is established explicitly with the call *tpconnect*. As long as this conversation is being held, the client and server can exchange data with *tpsend* and *tprecv*. The conversation is terminated when the server issues a *tpreturn*. The client then receives a corresponding code in the variable *tperrno* for *tprecv*. For this reason, the client program must contain at least one *tprecv* call.



svc identifies the local name of the service, *cd* the process-local communication descriptor, *tpservice* the program name of the service routine, and *svcinfo* the service info structure related to the service name and communication descriptor.

In this model, a dialog TAC must be generated on the UTM server side for the service.

In the event of an error, the client can abort a conversation with the call *tpdiscon*.

6.3 Typed buffers

XATMI applications exchange messages using "typed data buffers". This allows data passing through the network to be transferred correctly to the application, i.e. in accordance with the data structure and its data types underlying the buffer name.

This is advantageous in that the application need not consider machine dependencies such as big endian/little endian display, ASCII/EBCDIC conversion or alignment with word boundaries. This allows data types like *int*, *long*, *float* etc. to be transmitted as they are. The program no longer needs to perform any coding or decoding, as this is done by XATMI (in accordance with the rules of the XATMI U-ASE definition). A data buffer object consists of four components:

– Type: This defines the class of the buffer. There are three classes.

– Subtype: This defines the object of the class, i.e. the actual data structure.

– Length

– Data contents

Such a data buffer is created during runtime and can then be addressed via its variable name (=subtype name). In C programs, such buffers are created dynamically with *tpallcoc* and called “typed buffers”; in Cobol programs, these buffers are statically specified and called “typed records”.

Types

The type of data buffer specifies which elementary, language-dependent data types are permissible. This allows a common recognition of data in a heterogeneous client-server network. Three types are defined with XATMI:

X_OCTET	Untyped data flow of bytes ("user buffer"). This type does not have any subtypes. No conversion is carried out.
X_COMMON	All data types used jointly by C and COBOL. Conversion is carried out by XATMI.
X_C_TYPE	All elementary C data types except for pointers. Conversion is performed by XATMI.

Subtypes

Subtypes have a name up to 16 characters long, under which they are addressed in an application program. Every subtype is assigned a data structure (C structure or COBOL record) which specifies the syntax of the subtype, see page 105. Allocation between data structures, subtypes and required services is performed in the local configuration, see page 108.

The following table provides an overview of the elementary ASN.1 data types (basic types, see ISO 8824 [29]) and their codes.

Code[1]	Meaning	ASN.1 type	X_C_TYPE	X_COMMON
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	--[2]
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 to 255 (decimal)	OCTET STRING	char[n]	PIC X(n)
C!<n>	character array, terminated with zero ('\0')	OCTET STRING	char[n]	--
C<m>:<n>	character matrix[3]	SEQUENCE OF OCTET STRING	char [m][n]	--
C!<m>:<n>	character matrix, terminated with zero ('\0')	SEQUENCE OF OCTET STRING	char [m][n]	--
T<n>	The printable characters A-Z, a-z and 0-9 plus[4] a number of special characters and control characters	T.61-String	t61str[n]	PIC X(n)
T!<n>	character array, terminated with zero ('\0')t	T.61-String	t61str[n]	--
T<m>:<n>	character matrix	SEQUENCE OF T.61-String	t61str[m][n]	--
T!<m>:<n>	character matrix, terminated with zero ('\0')t	SEQUENCE OF T.61-String	t61str[m][n]	--

1 Serves to describe data structures in the local configuration

2 -- : not available in X_COMMON

3 a character matrix is a two-dimensional character array

4 in accordance with CCITT recommendation T.61 and. ISO 6937

Character set conversion with X_C_TYPE and X_COMMON

The data buffers are sent as ASCII code through the network. However, a partner can use a character set coding different to ASCII, such as a BS2000 application which uses EBCDIC. In this case, the XATMI library converts the ASN.1 character string type *T61String* (see ISO 8824 [32]) for all incoming and outgoing data. For this reason, automatic conversion must not be generated for the respective carrier system (openUTM/UPIC/OpenCPIC).

6.4 Program interface

The following sections provide an overview of the XATMI client program interface. A detailed description of the program interface as well as the error and return codes is contained in the X/Open specification for XATMI [28]. A knowledge of this specification is indispensable for the creation of XATMI programs.

The program interface is available for C as well as COBOL.

6.4.1 XATMI functions for clients

The following table lists all XATMI client calls as well as the two additional openUTM client calls, and describes the communications models for which they are used.

C call	COBOL call	Communications model
tpcall	TPCALL	Synchronous Request/Response
tpacall[1]	TPACALL	Asynchronous Request/Response
tpgetreply	TPGETRPLY	Asynchronous Request/Response
tpcancel	TPCANCEL	Asynchronous Request/Response
tpconnect	TPCONNECT	Conversational
tpsend	TPSEND	Conversational
tprecv	TPRECV	Conversational
tpdiscon	TPDISCON	Conversational
tpalloc	--	Request/Response and Conversational
tprealloc	--	Request/Response and Conversational
tpfree	--	Request/Response and Conversational
tptypes	--	Request/Response and Conversational
tpinit	TPINIT	Initialize client for UPIC carrier system; not contained in the XATMI standard
tpterm	TPTERM	Log client out of UPIC carrier system; not contained in the XATMI standard

[1] Single requests are only possible with a set *TPNOREPLY* flag for *tpacall*

The two functions *tpinit* and *tpterm* are not contained in the XATMI standard, and serve to connect XATMI to the carrier system. They are described in the following section.

In the case of the OpenCPIC carrier system, explicit login and logout with *tpinit / tpterm* is possible but not required. On no account must the user ID and/or password be transferred with the call *tpinit*.

6.4.2 Connection with the carrier system

The openUTM client uses UPIC or OpenCPIC as the carrier system. Consequently, an XATMI client program must explicitly log into the carrier system with *tpinit* and log out of it with *tpterm*:

1. *tpinit()*
2. XATMI calls, e.g. *tpalloc(), tpcall(), tpconnect(),...tpdiscon()*
3. *tpterm()*

The two calls *tpinit* and *tpterm* are described in the following; all other XATMI calls are described in the XATMI specification [28].

tpinit - initializing a client

Syntax

```
C:      #include <xatmi.h>

    int tpinit (TPCLTINFO *tpinfo)          (in)


COBOL: 01 TPCLTDEF-REC.
        COPY TPCLTDEF.

         CALL "TPINIT" USING TPCLTDEF-REC.
```

Description

The function *tpinit* initializes a client and identifies it to the carrier system. It must be called as the **first** XATMI function in a client program. In C, a pointer is to be transferred as a parameter to the predefined structure *TPCLTINFO*; in COBOL, the COBOL record *TPCLTDEF* must be supplied.

C structure *TPCLTINFO*:

```
#define  MAXTIDENT  9

typedef struct {
longflags;                    /* for future use */
charusrname[MAXTIDENT];
charcltname[MAXTIDENT];
charpasswd [MAXTIDENT];
} TPCLTINFO;
```

```
COBOL record TPCLTDEF:

05 FLAGPIC S(9) COMP-5.
05 USRNAMEPIC X(9).
05 CLTNAMEPIC X(9).
05 PASSWDPIC X(9).
```

A user ID is entered in *usrname* and a password is entered in *passwd*. Both parameters are used for setting up a conversation and serve to verify access authorization on the UTM side. *cltname* (= local client name) identifies the client for the carrier system.

cltname is the name in the LOCAPPL statement in OpenCPIC.

If *usrname* and *passwd* are initialized with the nullstring (COBOL: SPACES), then the security functions are not activated, i.e. access verification is not performed by openUTM. If at least one of these parameters contains a valid value, then this value is tested by openUTM. A nullstring in the parameter *cltname* implies a default setting of 8 blanks. If an empty *cltname* like this is specified, then the entry of a default server is sought in the OpenCIPIC generation file.

If *tpinit* is called up in C with a NULL pointer, then access verification is inactive and the "local client name" is reserved with 8 blanks. In COBOL, the structure needs to be supplied with SPACES for this purpose.

The entries in *usrname*, *passwd* and, if applicable, in *cltname* must conform with UTM notational conventions, i.e. they must be no longer than 8 characters and must be concluded with the end-of-string sign ("\0") in C.

Return values

tpinit returns -1 on the occurrence of an error, and sets the error variable *tperrno* to one of the following values:

TPEINVAL
: One or more parameters were supplied with an invalid value.

TPENOENT
: Initialization could not be performed, e.g. due to insufficient memory for internal buffers.

TPEPROTO
: *tpinit* was called in an impermissible situation, e.g. the client had already been initialized.

TPESYSTEM
: In internal error has occurred.

tpterm - logging out a client

Syntax

```
C:      int tpterm ()
COBOL: CALL "TPTERM".
```

Description

The function *tpterm* logs a client initialized with the last *tpinit* out of the carrier system. After *tpterm*, no more XATMI calls are permissible except for a renewed *tpinit*.

Return values

tpterm returns -1 on the occurrence of an error and sets the error variable *tperrno* to one of the following values:

TPENOENT
: The client could not log out correctly, e.g. due to a problem in the carrier system.

TPEPROTO
: *tpterm* was called in an impermissible situation, i.e. the client had not been initialized yet.

TPESYSTEM
: An internal error has occurred.

6.4.3 Include files and COPY elements

C modules with XATMI calls require the following include files:

1. The file *xatmi.h*.
2. The file(s) with the data structures for all typed buffers used in the module, also see page 97.

COBOL modules with XATMI calls require the following COPY elements:

1. TPSTATUS, TPTYPE, TPSVCDEF and TPCLTDEF.
2. The file(s) with the data structures for all typed records used in the module.

6.4.4 Events and error handling

On the occurrence of an event or error, XATMI functions return a value of -1. For precise determination of an event or error, the program needs to evaluate the variable *tperrno*.

In the conversational function *tprecv, tperrno=TPEEVENT* indicates that an event has occurred. This event can be determined by evaluating the *tprevc* parameter *revent*. Successful completion of a conversational service, for example, is indicated as follows:

```
Returncode from tprecv =-1
tperrno=TPEEVENT
revent=TPEV_SVCSUCC
```

For the function *tpsend,* the parameter *revent* has no significance.

When the service function is completed with *tpreturn,* the service program can use the parameter *rcode* to return a freely defined error code which can be evaluated on the client side via the external variable *tpurcode*, see the X/Open specification on XATMI [28].

6.4.5 Creating a typed buffer

Typed buffers are defined by data structures in include files (with C) and COPY elements (with COBOL), which need to be incorporated in the programs involved.

Data between the programs is exchanged in accordance with these data structures, which must therefore by known to the client as well as the server. All the data types described in the table on page 97 are permissible here.

The include and COPY files describing the typed buffers serve as entries for the generation program *xatmigen* (see page 111). The following rules apply to these files:

– C and COBOL data structures must be contained in separate files. Files containing a mixture of C include files and COBOL COPY elements are not permissible as entries.

– The files must only comprise the definitions of the data structures, blank spaces and commentaries. Macro statements i.e. those beginning with '#' are permissible in C.

– The data structure definitions must be complete. In particular, COBOL data records must begin with level number "01".

– The data structures must not be nested.

– Only absolute values - no macro constants - are permissible as field lengths

– Only the data types described in the table on page 97 are permissible. In particular: C pointer types are impermissible.

If necessary, the generation program *xatmigen* must be used to map the character arrays to the ASN.1 character string types (see ISO 8824 [29]), as neither C nor COBOL recognize these data types (see the section “xatmigen generation program” on page 111).

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

One simple example each for C and COBOL is shown in the following:

Example 1: C include for typed buffer

```
typedef struct {
charname[20];/* name of person */
intage; /* age */
charsex;
longshoesize;
} t_person;

struct t_city {
charname[32];/* name of city */
charcountry;
longinhabitants;
shortchurches[20];
longfounded;
}
```

Example 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.
```

Additional examples are provided in the X/Open specification on XATMI [28].

6.4.6 Characteristics of XATMI in openUTM client

This section describes the special aspects of implementing the XATMI interface in openUTM.

Supported XATMI calls

All XATMI calls relevant to clients are supported, in addition to the two calls *tpinit* and *tpterm*.

Limiting conditions

- For clients using the OpenCPIC carrier system, up to 256 conversations per service ar possible.
- A maximum of 100 buffer entities can be used simultaneously within a client application. In the case of a C application, for example, this implies a maximum of 100 *tpalloc* calls without a *tpfree* call.
- The maximum message length is 32000 bytes.

  The maximum size of a typed buffer is always smaller than the maximum possible message length, as the messages contain an overhead in addition to the net data. The more complex a buffer is, the higher the overhead. The rule of thumb here is: Maximum buffer size = 2/3 of the maximum message length.Consequently, the conversational model (*tpsend* / *tprecv*) should be used for larger amounts of data.
- The following limiting values apply to name lengths:

Service name	16 bytes
Buffer name	16 bytes

  In accordance with the standard, service names can be up to 32 bytes long, but only the first 16 are significant (constant *XATMI_SERVICE_NAME_LENGTH*). It is therefore advisable to restrict the service name to a maximum of 16 bytes.

Operation with the TX interface

When a service is called, the *TPTRAN* / *TPNOTRAN* flag is used to specify whether the service is included in the global transaction.

If a service is started with a set *TPTRAN* flag, it is included implicitly in the global transaction. No explicit TX calls are required in the program.

In the case of *tpreturn, TPSUCCESS* and *TPFAIL* are used respectively to specify whether the transaction is committed or rolled back.

For more details, refer to the X/Open specification on the XATMI interface [28] (chapter titled “Transaction Functions Affecting the XATMI Interface”).

6.5 Configuration

For every XATMI application, the user needs to generate a local configuration. This configuration describes the available and utilized services and their destination addresses, as well as the utilized typed buffers and their syntax. The information is stored in the local configuration file (LCF), which is read once by the application on starting. An LCF is required for the client side and the service side.

6.5.1 Creating a local configuration file

The user needs to create an input file, or LC definition file. This input file must consist of individual lines whose syntax is governed by the following rules:

– A line begins with an SVCU or BUFFER statement and specifies exactly one service or subtype (=typed buffer).

– Two operands are separated by a comma.

– A statement line is concluded with a semicolon (';').

– If the operands occupy more than one line, then each line must be concluded with a continuation symbol '\' (backslash).

– A commentary line begins with a '#' symbol in the first column.

– Blank lines can be inserted to improve readability.

The local configuration file (page 111) is created from the LC definition file with the help of the generation program *xatmigen*.

The SVCU and BUFFER statements are described in the following.

SVCU statement: Defining an addressable service

An SVCU statement describes to a client the properties required for calling up a service in the partner application.

Operator	Operands	Explanation
SVCU	internal-service-name	Mandatory operand, maximum 16 bytes
	[,RSN=remote-service-name]	Standard: internal service name
	[,TAC=transaction-code]	Standard: internal service name
	,{DEST=destination-name | .DEFAULT }	Mandatory operand, partner application
	[,MODE=[RR |CV]]	Standard:RR (Request/Response)
	[,BUFFERS=(subtype-1,...,subtype-n)]	Standard: no subtype

internal-service name
: Name with a maximum length of 16 bytes, under which a (remote) service is addressed in the program. This name must be unique within the application, i.e. it must only occur once in the LCF.

  Multiple definitions are not checked. The first *internal-service-name* is valid; any further identical names are ignored.

  Mandatory operand!

RSN=remote-service-name
: Name - maximum length 16 bytes - of a service in a *remote* application. This name is transferred to the remote application. It can occur repeatedly in the LCF.

  If this operand is omitted, then RSN=*internal-service-name* is set by default.

TAC=transaction code
: Transaction code with a maximum length of 8 bytes, under which the service in the remote application must be generated.

  If this operand is omitted, then *xatmigen* sets TAC=*internal-service-name* by default and shortens it to the first 8 bytes if necessary.

  The transaction code *KDCRECVR* can be used to define a recovery service which sends the last output message from openUTM to the client.

DEST=destination-name
: Identification of the partner application (maximum length of 8 bytes).

  This name must be generated in a SYMDEST statement of the OpenCPIC generation file (see page 42).

  .DEFAULT
  A default server is used.

MODE
: This determines which communication model is used for the service:

  = RR Request response model (default value)
  = CV Conversational model

BUFFERS=(subtype-1,...,subtype-n**)**
: This is a list of subtype names which can be sent to the service. Every name has a maximum length of 16 bytes; all the characters of the ASN.1 character string type *PrintableString* (refer to the section “Character sets” on page 191 and ISO 8824 [29]) are permissible.

  For every subtype listed here, a separate BUFFER statement defining the characteristics of the subtype must be specified (see below).

The BUFFERS operand is position-sensitive and must (if included) always be the last operand of a statement.

If BUFFERS is omitted, then only a buffer of type *X_OCTET* should be sent to the service.

BUFFER statement: Defining a typed buffer

A BUFFER statement defines a typed buffer. Buffers with identical names must be defined identically on the client and server sides.
Multiple definitions are not checked. The first buffer entry is valid, all others are ignored.

Buffers of the type *X_OCTET* have no particular characteristics and thus do not need to be defined. Typed buffers are defined with the following parameters:

Operator	Operands	Explanation
BUFFER	subtype-name	Maximum 16 bytes
	[,REC=referenced-record-name]	Standard: subtype-name
	[,TYPE=X_COMMON | X_C_TYPE]	Standard: xatmigen sets TYPE automatically

subtype-name
: This is the name of the buffer - with a maximum length of 16 bytes - and must also be specified in the BUFFERS operand of the SVCU statement. The name must be unique in the application.

REC=referenced-record-name
: Name of the data structure for the buffer; in the case of C structures, for example, this is the "typedef" or "struct" name.

  If this operand is omitted, then *xatmigen* sets REC=*subtype-name*.

TYPE=
: Buffer type; more details on types are provided on page 97.

  If this operand is omitted, then xatmigen sets the type to *X_C_TYPE* or *X_COMMON*, depending on which elementary data types were used.

  TYPE is ignored if X_COMMON or X_C_TYPE has not been set or if the data structure is not related to the specified buffer type.

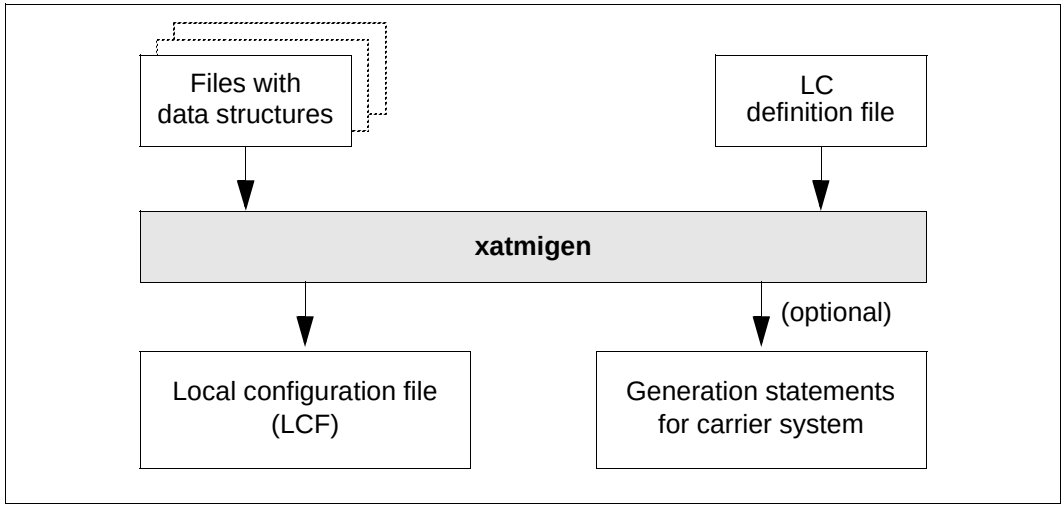
During generation, *xatmigen* creates two additional operands with the following meanings:

LEN=length — Length of the data buffer.

SYNTAX=code — Syntax description of the data structure in code representation, as listed in the table on page 97.

6.5.2 xatmigen generation program

xatmigen prepares a local configuration file (LCF) from a file containing the definition of the local configuration (LC definitions file) and one or more files with C or COBOL data structures (LC description files) as shown in the diagram below:



The local configuration file is structured just like the LC definition file, and only differs from it in terms of the description of the buffer type, buffer length and buffer syntax string. In other words, the BUFFER statements are extended with respect to the LC definition file by the operands LEN, SYNTAX and, if applicable, TYPE.

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. A file containing the generation statements for the carrier system can be created optionally.

Success and failure messages are written to *stdout* and *stderr*.

Although it is possible to edit the LCF, we strongly advise against this.

The service program *xatmigen* is located in the path *instpfad/upic/xatmi/ex/xatmigen*.

Syntax

xatmigen [system] **-d** lcdf_name [**-l** lcf_name] [**-i**] [**-c** stringcode] [descript_file_1] ... [descript_file_n]

Description

system

If specified, a file containing generation statements for the carrier system *system* is created. Permissible entries here are:

upic Entries for upicfile are created (file name: *xtupic.def*)

cpic A fragment is created for OpenCPIC generation. This fragment needs to be extended (file name:*xtcpic.def*)

If it has been specified, *system* must always be the first parameter of *xatmigen*. If *system* is not specified, then no generation statements are created.

-d lcdf_name

Name of the LC definition file. This switch must always be specified.

-l lcf_name

Name of the local configuration file to be created. This name must conform with the conventions of the operating system being used.

Any existing LCF of the same name is overwritten without commentary.

If the switch is omitted, then *xatmigen* creates the file *xatmilcf* in the current directory.

-i Interactive mode, i.e. for every typed buffer containing a character array, the related string code is scanned. The permissible entries for the string code are described under switch *-c*.

Switch *-i* has precedence over switch *-c*. When *xatmigen* is running in batch mode, switch *-i* must not be included.

-c stringcode

The specified string type *stringcode* applies to the entire *xatmigen* process, i.e. for all character arrays. Switch *-c* is ignored in the interactive mode (switch *-i*).

The following entries are permissible for *stringcode* (refer to the table on page 97):

C|c	*OCTET STRING*
C!|c!	Null-terminated *OCTET STRING*
T|t	*T61String*
T!|t!	Null-terminated *T61String* (standard value)

If no entry has been made, the default value *T!* is set.

descript_file_1 . . . descript_file_n
: List of files containing the include and COPY elements with the data structures of the typed buffers.

  If the entry is missing, only buffer type X_OCTET is permissible.

Note

Switch **-d** and, if included, switches **-l** and **-c** must be followed by the related parameters. It is not permissible to include a switch without the corresponding parameter.

6.5.3 Configuring the carrier system OpenCPIC and UTM partners

To make an XATMI application in OpenCPIC functional, you must

– match OpenCPIC generation with the local configuration and partner generation,

– match the initialization parameters specified in *tpinit* with the generation of the UTM application.

6.5.3.1 Generating OpenCPIC

An XATMI client with OpenCPIC as the carrier system is generated just as a pure OpenCPIC application (see the chapter “Generating OpenCPIC” on page 29). The following special aspects must be taken into account here:

– The *locappl_name* in the LOCAPPL statement (see page 32) is transferred on the call *tpinit*.

– The *symdest_name* in the SYMDEST statement (see page 42) must coincide with the value of the DEST operand in the SVCU statement (see page 108). PARTNER-APRO=XATMI must be set additionally.

6.5.3.2 Configuring UTM partners

An OSI-TP coupling is to be generated on the UTM side for the KDCDEF process.

Example:

```
ACCESS-POINT SERVACCP,
             P-SEL=NONE, S-SEL=NONE, T-SEL=C'SERVNAME'

OSI-LPAP RCLTNAME,
         ASSOCIATION-NAMES = ASSONAME,
         ASSOCIATIONS = 2,
         CONTWIN = 1,
         CONNECT = 1,
         PERMIT = ADMIN

 OSI-CON RCLTNAME,
         P-SEL=NONE, S-SEL=NONE,
         T-SEL=C'LCLTNAME',
         N-SEL=C'CLTNODE',
         LOCAL-ACCESS-POINT = SERVACCP,
         OSI-LPAP = RCLTNAME,
         ACTIVE = YES
```

If the client and server are running on different systems, a corresponding TNSX generation is also required.

6.5.3.3 Initialization parameters and UTM generation

An XATMI client is initialized with the call *tpinit*. Parameters for the user ID, password and local name are transferred in the structure *TPCLTINIT*.

In case of the OpenCPIC carrier system, the name generated in TNSX must always be specified for the local name parameter. The TNS name must then be defined with the parameter TSEL in the OSI-CON statement on the UTM side.

6.6 Linking and starting XATMI client programs

On the linkage of an XATMI client program, the following libraries must be linked together with the user modules:

– XATMI client library *libxtclt.so* (path *instpfad/upic/xatmi/sys/*)
– CPI-C library *openCPIC-path/libocpic.a*
– Unix/Linux system library “mathlib” (switch *-lm*)

An XATMI client program is started as an executable program, just as other OpenCPIC application programs (see the section “Starting application programs” on page 86). The OpenCPIC manager must have been started previously.

6.6.1 Environment variables

Before an XATMI client program is started, the following environment variables must be set:

XTCLTTR This environment variable must be set in order to activate the XATMI trace (see section “Activating XATMI trace” on page 116).

XTPATH This is the path name for the log files of the XATMI trace (see section “Activating XATMI trace” on page 116).

XTLCF File name for the local configuration file (LCF).
If this variable is not set, then a file with the name *xatmilcf* is sought in the current working directory.

6.6.2 Activating XATMI trace

For diagnostic purposes, you can activate the trace of the library *libxtclt.a*. As a result, *libxtclt.a* records trace information while the program is running. The XATMI trace operates in two (trace levels): interface trace (level 1) and error trace (level 2). For error diagnosis in XATMI application programs, the interface trace is generally sufficient.

The XATMI trace is controlled via environment variables. The following environment variables are evaluated by the library:

XTCLTTR This environment variable must be set in order to activate the XATMI trace.

- **=E** (Error)
  This activates the error trace.
- **=I** (Interface)
  This activates the interface trace for the XATMI calls.
- **=F** (Full)
  This activates the error trace and the interface trace.

XTPATH=pathname This indicates the directory in which the log files of the XATMI trace are to be stored. If *XTPATH* is not set, the log files are stored in the current working directory of the XATMI client program.

Every XATMI client process writes the trace information to a separate file which can exist in two generations (old and new).

The maximum size of a log file is roughly 128 kbytes. Once this size has been attained, a change is made to a second file. Once this file has also attained the maximum size, the first one is overwritten again. The name of a log file of an XATMI client program is composed of the following elements:

XTCpid**.**suff

pid and suff are placeholders with the following meaning:

pid Process ID of the XATMI client process

suff File name suffix indicating the generation: 1 or 2
The more recent log file can be determined from the time stamp.

The log files are in ASCII format and can be read directly without any further editing.

7 Communicating with partner applications

The program interfaces X/Open CPI-C 2.0 and X/Open TX of the OpenCPIC product allow you to communicate with partners accessible via OSI-TP and perform global transactions.

This chapter provides examples and information on possible configurations in various client-server environments.

A partner application of OpenCPIC can be:

– OpenCPIC
– a UTM application
– an OSI-TP application

A sample configuration for communication with an OpenCPIC application is provided in the section “OpenCPIC communication in a homogeneous environment” on page 118.

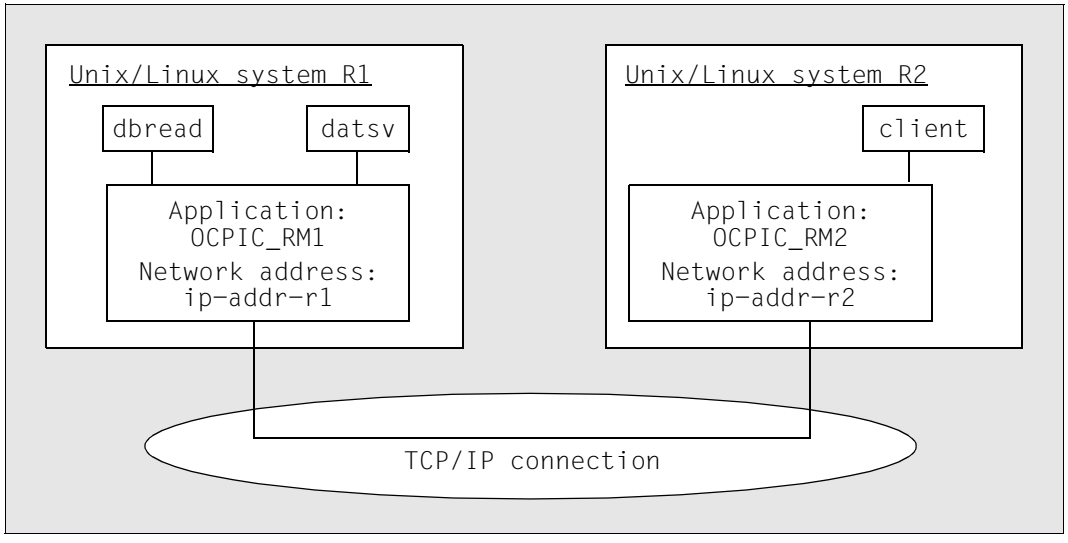
Communication with UTM applications is described in the section “OpenCPIC communication with openUTM” on page 121.

Notes on linkage with the applications of different manufacturers are contained in the section “OpenCPIC communication with third party systems” on page 135.

7.1 OpenCPIC communication in a homogeneous environment

This section provides an example of simple communication between two OpenCPIC application programs on different Unix or Linux systems.

The following diagram shows the configuration here:



Example of communication between two Open CPIC applications

OpenCPIC is installed and an OpenCPIC manager is configured on each of the two Unix or Linux systems R1 and R2. Two CPI-C application programs *dbread* and *datsv* are present on system R1. One application program *client* is present on system R2.

dbread

If required, the program */home/dbserver/dbread* is to be started by the OpenCPIC manager with the user ID *dbuser*. It supplies the partner with data from a local database. To improve the overview, the local resource manager was left out of the picture. It is of no significance for this configuration.) The partner is to address the program with the local name (TPSU title) *GETDB*.

The program */home/dbserver/dbread* contains the following CPI-C calls in the specified sequence:

```
Accept_Conversation
Receive
Receive
Send_Data
Deallocate
```

datsv

The program */home/cpic/datsv* is to be activated on every system start. It is only to receive data and store it in the local file system without interacting with a local resource manager.
The partner is to address this program with the local name (TPSU title) *RCVDATA*.

The program */home/cpic/datsv* contains the following CPI-C calls in the specified sequence:

```
Specify_Local_TP_Name          /* tp_name = RCVDATA */

in a loop:
Accept_Conversation
Receive
```

client

The CPI-C application program */home/locpappl/client* on system R2 is called up by different users. In accordance with requirements, it is to set up conversations with the program *GETDB* (*dbread*) or *RCVDATA* (*datsv*) on system R1.

The program */home/locpappl/client* contains the following CPI-C calls:

```
if ( ... )                    /* Required partner GETDB */
{
    Initialize_Conversation   /* sym_dest_name = GETDB  */
    Allocate
    Send_Data
    Prepare_To_Receive
    Receive
}
else                          /*Required partner RCVDATA */
{
    Initialize_Conversation   /* sym_dest_name = RCVDATA */
    Allocate
    Send_Data
    Deallocate
}
```

TNSX and OpenCPIC generation as well as the RC file for OpenCPIC in this example are shown below:

TNSX generation (tnsxcom**)**

System R1

```
; local OpenCPIC manager
ocpic_loc \
    PSEL V ''
    SSEL V ''
    TSEL LANINET A'102'
    TSEL RFC1006 A'ocp1'

; partner OpenCPIC manager
ocpic_r2 \
    PSEL V ''
    SSEL V ''
    TA RFC1006 ip-addr-r2 \
        PORT 102 A'ocp2'
```

System R2

```
; local OpenCPIC manager
ocpic_loc \
    PSEL V ''
    SSEL V ''
    TSEL LANINET A'102'
    TSEL RFC1006 A'ocp2'

; partner OpenCPIC manager
ocpic_r1 \
    PSEL V ''
    SSEL V ''
    TA RFC1006 ip-addr-r1 \
        PORT 102 A'ocp1'
```

OpenCPIC generation (ocpic_gen**)**

System R1

```
LOCAPPL ocpic_loc,
    APT = (1,17, 25, 1001),
    AEQ = 300

PARTAPPL ocpic_r2,
    APT = (1, 20, 9, 0, 0),
    AEQ = 12,
    CCR = YES

LOCAPRO GETDB,
    PATH = /home/dbserver/dbread,
    LOGIN = dbuser
```

System R2

```
LOCAPPL ocpic_loc,
    APT = (1, 20, 9, 0, 0),
    AEQ = 12

PARTAPPL ocpic_r1,
    APT = ( 1, 17, 25, 1001),
    AEQ = 300,
    CCR = YES

SYMDEST R1GETDBS,
    PARTNER-APPL = ocpic_r1,
    PARTNER-APRO = GETDB

SYMDEST R1RCVDAT,
    PARTNER-APPL = ocpic_r1,
    PARTNER-APRO = RCVDATA
```

System-RC file (/etc/rc2.d/S90OCPIC**)**

System R1

```
ocpic_start &
/home/ocpappl/datsv &
```

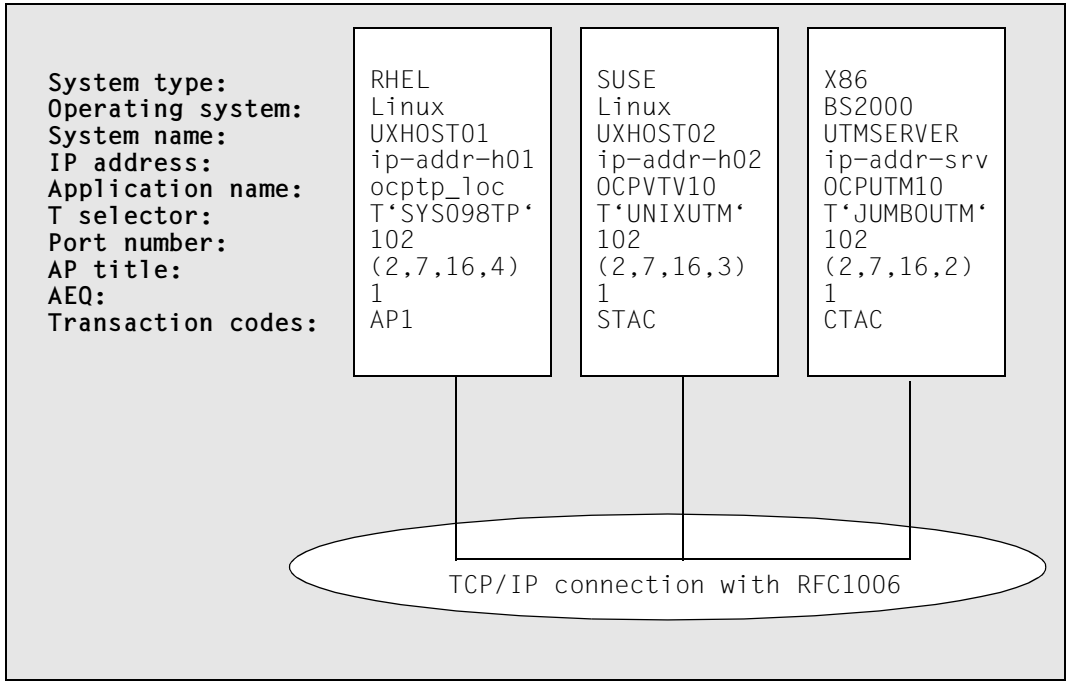
System R2

```
ocpic_start &
```

7.2 OpenCPIC communication with openUTM

This section provides an example of communication between OpenCPIC application programs on a Uniux or Linux system and UTM application programs on a BS2000 system or Unix/Linux system.

The following diagram shows this configuration:



Example of communication between an OpenCPIC and a UTM application

OpenCPIC generation, UTM generation, entries for the transport system and a number of sample programs for all systems are described in the following section.

UTM generation (KDCDEF) in the BS2000 system

```
MAX APPLINAME = OCPUTMAP,
UTMD APT = (2, 7, 16, 2),
ACCESS-POINT OCPUTM10,
    P-SEL = *NONE,
    S-SEL = *NONE,
    T-SEL = C'JUMBOUTM',
    AEQ = 1
OSI-CON S098TP80,
    P-SEL = *NONE,
    S-SEL = *NONE,
    T-SEL = C'SYS098TP',
    N-SEL = C'UXHOST01',
    LOCAL-ACCESS-POINT = OCPUTM10,
    OSI-LPAP = OLP80
OSI-LPAP OLP80,
    APT = (2,7,16,4),
    AEQ = 1,
    ASS-NAMES = SES1,
    ASSOCIATIONS = 10,
    CONTWIN = 5
LTAC CPICTP,
    LPAP = OLP80,
    RTAC = AP1
TAC CTAC ...
```

BCIN command in the BS2000 system:

```
/BCIN UXHOST01,IPADR=(ip-addr-h01),...
```

UTM generation (KDCDEF) in Linux system SUSE

```
MAX APPLINAME = utm608ap,
UTMD APT = (2, 7, 16, 3),
ACCESS-POINT OCPVTV10,
    P-SEL = *NONE,
    S-SEL = *NONE,
    T-SEL = C'utm608ot',
    AEQ = 1
OSI-CON S098TP81,
    P-SEL = *NONE,
    S-SEL = *NONE,
    T-SEL = C'sys098qs',
    N-SEL = C'UXHOST01',
    LOCAL-ACCESS-POINT = OCPVTV10,
    OSI-LPAP = OLP81
OSI-LPAP OLP81,
    APT = (2,7,16,4),
    AEQ = 1,
    ASS-NAMES = SES1,
    ASSICIATIONS = 10,
    CONTWIN = 5
LTAC CPICTP,
    LPAP = OLP81,
    RTAC = AP1
TAC STAC ...
```

TNSX generation for UTM in Linux system SUSE with tnsxcom

```
utm608ap\
        TSEL RFC1006 A'13659'
utm608ot\
        TSEL RCF1006 T'UNIXUTM'
sys098qs.UXHOST01\
        TA   RFC1006 ip-addr-h01 PORT 102 T'SYS098TP'
```

OpenCPIC generation in Linux system RHEL with ocpic_gen

```
LOCAPPL ocptp_loc,

    APT = (2,7,16,4),
    AEQ = 1
PARTAPPL ocpvtv10,
    APT = (2,7,16,3),
    AEQ = 1,
    ASSOCIATIONS = 10,
    CONTWIN = (5 5),
    CONNECT = 10
PARTAPPL ocputm10,
    APT = (2,7,16,2),
    AEQ = 1,
    ASSOCIATIONS = 10,
    CONTWIN = (5 5),
    CONNECT = 10
SYMDEST RUTMAC,
    PARTNER-APPL = ocpvtv10,
    PARTNER-APRO = STAC
SYMDEST BUTMAC,
    PARTNER-APPL = ocputm10,
    PARTNER-APRO = CTAC
LOCAPRO AP1
    PATH  = /home/server/ap1
    LOGIN = utmclien
```

TNSX generation for OpenCPIC in Linux system RHELwith tnsxcom

```
ocptp_loc\
        PSEL V''
        SSEL V''
        TSEL RCF1006 T'SYS098TP'
ocpvtv10\
        PSEL V''
        SSEL V''
        TA   RFC1006 ip-addr-h02 PORT 102 T'UNIXUTM'
ocputm10\
         PSEL V''
         SSEL V''
         TA   RFC1006 ip-addr-srv PORT 102 T'JUMBOUTM'
```

Sample programs

Example 1:

Single-step procedure with openUTM

```
Initialize_Conversation
Allocate
Send_Data
Receive                                 ->  INIT
                                            MGET
                                            MPUT
. . . CM_COMPLETE_DATA_RECEIVED         <-  PEND FI
Receive CM_DEALLOCATED_NORMAL
```

Example 2:

Multi-step procedure with openUTM, transmission of two consecutive messages

```
Initialize_Conversation
Allocate
Send_Data
Send_Data
Receive                                 ->  INIT
                                            MGET
                                            MGET
                                            MPUT
                                            MPUT
. . . CM_COMPLETE_DATA_RECEIVED         <-  PEND
Receive CM_COMPLETE_DATA_RECEIVED           RE/KP
Receive CM_SEND_RECEIVED
Send_Data
Receive                                 ->
                                            INIT
                                            MGET
. . . CM_COMPLETE_DATA_RECEIVED         <-  MPUT
Receive CM_DEALLOCATED_NORMAL               PEND FI
```

Example 3:

Asynchronous process with openUTM, acknowledgement on dialog completion

```
Initialize_Conversation
Set_Sync_Level CM_CONFIRM
Allocate
Set_Send_Type CM_BUFFER_DATA
Send_Data
Send_Data
Deallocate                                  --->  INIT
                                                  FGET
                                                  FGET
. . . CM_OK                                 <---  PEND FI
```

Example 4:

Invocation of a CPI-C application program as an asynchronous program by openUTM, a CPI-C application program has already been started.

```
Specify_Local_TP_Name
Initialize_For_Incoming
Accept_Incoming
                                                  APRO AM  > A
                                                  FPUT NT  > A
                                                  FPUT NT  > A
. . . CM_OK                                 <---  PEND FI
Receive CM_COMPLETE_DATA_RECEIVED
Receive CM_COMPLETE_DATA_RECEIVED
Receive CM_CONFIRM_DEALLOC_RECEIVED
```

Example 5:

Invocation of a CPI-C application program as a dialog program by openUTM, a CPI-C application program has not been started yet.

```
                                        APRO DM  > A
                                        MPUT NT  > A
Start by OpenCPIC manager             <---  PEND KP
Accept_Conversation
Receive CM_COMPLETE_DATA_RECEIVED
Send_Data
Deallocate                            --->  INIT
                                        MGET NT  > A  KCRST=(C,U)
```

Example 6:

A CPI-C application program requests an acknowledgement, a UTM application supplies a negative acknowledgement

```
Initialize_Conversation
Set_Sync_Level CM_CONFIRM
Allocate
Set_Send_Type CM_BUFFER_DATA
Send_Data
Prepare_To_Receive                    -->   INIT
                                            MGET
                                            MGET          KCRMGT=H
                                            MPUT EM
. . . CM_PROGRAM_ERROR_PURGING        <--   PEND FI
Deallocate
```

Example 7:

A UTM application requests an acknowledgement, CPI-C application provides a positive acknowledgement, a CPI-C application has already been started

```
Specify_Local_TP_Name
Initialize_For_Incoming
Accept_Incoming
                                          INIT
                                          MGET
                                          APRO DM  > A
                                          MPUT NT  > A
                                          MPUT HM  > A
. . . CM_OK                          <--  PEND KP
Receive CM_COMPLETE_DATA_RECEIVED
Receive CM_CONFIRM_SEND_RECEIVED
Confirmed
Send_Data
Deallocate                           -->  INIT
                                          MGET NT  > A  KCRMGT=C
                                          MGET NT  > A  KCRST=(C,U)
                                          MPUT NE
                                          PEND FI
```

Example 8:

Transmission of an error message by a CPI-C application program to the UTM application

```
Initialize_Conversation
Allocate
Send_Data
Send_Error
Send_Data
Prepare_To_Receive                   -->   INIT
                                           MGET
                                           MGET         KCRMGT=E
                                           MGET
                                           MPUT NE
Receive CM_COMPLETE_DATA_RECEIVED    <---  PEND FI
Receive CM_DEALLOCATED_NORMAL
```

Example 9:

Transmission of an error message from a UTM application to a CPI-C application program

```
Initialize_Conversation
Allocate
Send_Data
Prepare_To_Receive                      -->   INIT
                                              MGET
                                              MPUT
                                              MPUT EM
Receive CM_COMPLETE_DATA_RECEIVED       <---  PEND FI
Receive CM_PROGRAM_ERROR_PURGING
Receive CM_DEALLOCATED_NORMAL
```

Example 10:

Abnormal dialog termination by a CPI-C application program, a CPI-C application program has already been started

```
Specify_Local_TP_Name
Initialize_For_Incoming
Accept_Incoming
                                              INIT
                                              MGET
                                              APRO DM  >A
                                              MPUT NT  >A
. . . CM_OK                              <---  PEND KP
Receive CM_COMPLETE_DATA_RECEIVED
Set_Deallocate_Type CM_DEALLOCATE_ABEND
Deallocate                               --->  INIT
                                              MGET NT  >   KCRST=(Z,U)
                                              MPUT NE  A
                                              PEND FI
```

Example 11:

Abnormal dialog termination by the UTM application

```
Initialize_Conversation
Allocate
Send_Data
Prepare_To_Receive                    -->   INIT
                                            MGET
                                            MPUT
Receive CM_DEALLOCATED_ABEND          <---  PEND FR
```

Example 12:

Abnormal dialog termination by the UTM application without process cancellation

```
                                            INIT
                                            MGET
                                            APRO DM  > A
                                            MPUT NE  > A
Accept_Conversation                   <---  PEND KP
Receive CM_COMPLETE_DATA_RECEIVED
Receive CM_SEND_RECEIVED
Send_Data                             -->   INIT
                                            MGET     > A
                                            CTRL AB  > A
                                            MPUT NE
Receive CM_DEALLOCATED_ABEND          <---  PEND FI
```

Example 13:

Committing a transaction, OpenCPIC as client

```
tx_open
tx_begin
Initialize_Conversation
Set_Sync_Level CM_SYNC_POINT
Allocate
Send_Data
Receive                              --->   INIT PU
                                                          KCENDTA=O
                                                          KCSEND=Y
                                            MGET          KCRST=
                                            MPUT NT       (0,0)
. . . CM_COMPLETE_DATA_RECEIVED      <---   PEND KP
Send_Data
Deallocate
tx_commit                            --->   INIT PU

                                                          KCENDTA=F
                                            MGET          KCSEND=N
. . . TX_OK                          <---   PEND FI        KCRST=(0,P)
tx_close
```

Note: If the second UTM application program ends with *PEND FR* instead of *PEND FI*, the transaction is rolled back. In this case, the OpenCPIC application program receives the return value *TX_ROLLBACK* with *tx_commit()*.

Example 14:

Two chained transactions, OpenCPIC as server

```
                                              INIT
                                              MGET
                                              APRO DM  > A KCOF=C
                                              MPUT NT  > A
                                              CTRL PR  > A
tx_open                                <---   PEND KP
Accept_Conversation
Receive
Receive CM_TAKE_COMMIT__DATA_OK
Send_Data
tx_commit                              --->   INIT
                                              MGET NT  > A KCRST=(O,P)
                                              MPUT NE
                                              PEND RE

                                              INIT
                                              MGET
                                              MPUT NT  > A
                                              CTRL PE  > A
Receive                                <--    PEND KP
Receive
CM_TAKE_COMMIT_DEALLOCATE_DATA_OK
Send_Data
tx_set_transaction_control TX_UNCHAINED -->   INIT
tx_commit                                     MGET NT  > A KCRST=(C,P)
tx_close                                      MPUT NE  > A
                                              PEND FI
```

Example 15:

Committing a transaction with transfer of transmission rights, OpenCPIC as client

```
tx_open
tx_begin
Initialize_Conversation
Set_Sync_Level CM_SYNC_POINT
Allocate
Send_Data
Prepare_To_Receive
tx_commit                              --->  INIT PU
                                                        KCENDTA=C
                                                        KCSEND=Y
                                             MGET       KCRST=(O,P)
                                             MPUT NT
. . . TX_OK                            <---  PEND RE
Receive CM_COMPLETE_DATA_RECEIVED
Receive CM_SEND_RECEIVED
... the program can be continued as
in example 13
```

Example 16:

Access control with user ID and password

```
Initialize_Conversation “SECU0001”
Set_Conversation_Security_Type
CM_SECURITY_PROGRAM
Set_Conversation_Security_User_ID
“CPIC1”
Set_Conversation_Security_Password
“12345678”                             --->  INIT
Allocate                                     MGET
Send_Data                                    NPUT NE
Receive                                <---  PEND FI


. . . CM_COMPLETE_DATA_RECEIVED
Receive CM_DEALLOCATED_NORMAL
```

The following special generations are required in this case:

:

UTM generation (KDCDEF

```
. . .
ACCESS-POINT . . .
OSI-CON . . . ,
    OSI-LPAP = OPCPIC1, . . .
OSI-LPAP OPCPIC1,
    APPLICATION-CONTEXT = UDTSEC
TAC SECU1, PROGRAM = . . .
USER CPIC1, PASS = 12345678
. . .
```

OpenCPIC generation (*ocpic_gen*)

```
PARTAPPL UTMAW,
    APT = . . . ,
    APPL-CONTEXT = utm-secu

SYMDEST SECU0001,
    PARTNER-APPL = UTMAW,
    PARTNER-APRO = SECU1
```

7.3 OpenCPIC communication with third party systems

In principle, communication between OpenCPIC and applications from third party manufacturers is possible if the OSI-TP protocol is used. In this case, however, you must precisely determine the extent to which OSI-TP is supported by the third party system.

In OpenCPIC, the OSI-TP protocol elements are mapped to CPI-C and vice versa as described in the X/Open specification of the interface CPI-C 2.0 (appendix D).

Note that application programs on third party systems can be implemented on a communication interface completely different to CPI-C, e.g. CICS-API (customer information control system - application programming interface).

The following items are of significance for heterogeneous links with OpenCPIC:

1. The third party system must implement at least the OSI-TP functional units *Dialogue* and *Polarized Control*. Communication with applications which only offer *Dialogue* and *Shared Control* is not possible.
   OpenCPIC also supports the functional units *Handshake*, *Commit*, *Chained transactions* and *Unchained transactions*. If they need to be used, the partner must support them too.
2. OSI-TP assumes that the applications can specify any data structure by selecting a U-ASE (user application service element) during the establishment of an association. OpenCPIC application programs can only specify the standard scope of the U-ASE UDT (unstructured data transfer, ISO-10026-6, [32]). User data is transmitted as *Octet-String* with *UD-TRANSFER-RI*. The data structures to be used must be agreed privately between the application programs.
3. If the partner program is to transmit and receive all data in EBCDIC representation, although the local application program only uses the ASCII character set, then the CPI-C calls *Convert_Incoming (CMCNVI)* and *Convert_Outgoing (CMCNVO)* can be used for data conversion. The conversion tables are shown in the appendix on page 194.
4. *UDT* does not specify a maximum length for user data. With OpenCPIC however, only data of length 0 to 32767 bytes (32 KB) can be transmitted and received. If data longer than 32 KB is received from a partner program, the association to this partner is terminated.
5. OSI-TP allows the transmission of user data during dialog setup using *TP-BEGIN-DIALOGUE-request* and *TP-BEGIN-DIALOGUE-confirm* (initialization data) and during dialog termination using *TP-ABORT-request* (log data). OpenCPIC does not support this, i.e. an OpenCPIC application program cannot send initialization or log data, and incoming data is lost.
6. The TPSU titles must be selected on both sides in such a manner that they can be called up by the respective partner.

In OpenCPIC a remote TPSU title (partner AP name) can be defined either through generation with the SYMDEST statement (parameters PARTNER-APRO and PARTNER-TYPE) or with the CPI-C call *Set_partner_TP_Name (CMSTPN)*. The SYMDEST statement permits TPSU titles for the type *PrintableString*, *T61String* and *Integer*. If the TPSU title is specified with *CMSTPN*, it automatically receives the type *T61String*.

A local TPSU title (local AP name) can be defined either through generation with the LOCAPRO statement or with the CPI-C call S*pecify_Local_TP_Name (CMSLTP)*. In both cases, the TPSU title consists of a character string with a maximum length of 64 characters. On reception of a *TP-BEGIN-DIALOGUE-indication*, OpenCPIC therefore only accepts TPSU titles of the type *PrintableString* and *T61String* with a maximum length of 64. Dialog requests for a local TPSU title of the type *Integer* are rejected by the OpenCPIC manager.

7. OSI-TP allows several APDUs (application protocol data units) to be concatenated to a PSDU (presentation service data unit). In principle, this allows PSDUs of any required length to be created. However, OpenCPIC can only process PSDUs with a length of up to 33 KB. If longer PSDUs are received, the link to the partner is closed down. Accordingly, OpenCPIC also transmits PSDUs with a length of up to 33 KB.

   Thus, an OSI-TP implementation on third party systems must transmit PSDUs no longer than 33 KB, and be able to process incoming PSDUs with a length of up to 33 KB during communications with OpenCPIC.

8 Global transactions

This chapter describes the characteristics and functions of OpenCPIC which are of special significance for global transactions. A detailed definition of the global transaction model is provided in the X/Open documents ([24],[24],[25],[26]).

8.1 Local resource managers - XA connection

Local resource managers control access and modifications to local resources such as file systems and databases. To perform global transactions in accordance with the X/Open DTP reference model, all the involved resource managers need to be able to handle the two-phase commit protocol. A resource manager should be capable of undoing all modifications to local resources up to a certain stage of any transaction. The modifications are to be saved only once a confirmation has been sent by all the communication partners involved in the transaction.

X/Open has defined the interface XA for this purpose. The transaction manager (TM) and the TX library in OpenCPIC use this interface to communicate with a local resource manager.

The implementation of the XA functions is normally part of the resource manager and usually available in the form of one or more libraries. These libraries must be linked with the OpenCPIC manager to allow the TM to access the local resource manager. This procedure is described in the following. In addition, the XA libraries must be linked with the local application programs which need to access local resources via the local resource manager during transactions. This procedure is described in the section titled “Linking application programs“ on page 85.

In principle, the X/Open DTP reference model allows an application to communicate with several local resource managers simultaneously. At present however, OpenCPIC only supports one resource manager per application.

The currently supported database products are listed in the release notice.

The *openCPIC-path/xa/ocpic_xa_connect* file contains the connection for the XA interface in the form of a dynamic library. This file must contain the XA connection for the resource manager currently being used. *ocpic_xa_connect* is loaded dynamically by *ocpic_mgr* at runtime in order to address the resource manager.

You must save the open string for the database in the *openCPIC-path* directory in a file with the name *op.<XA-switch-name>*. Please take the XA switch name for the database concerned from the *openCIPC-path/xa/xa_info.c* file. For the Oracle database, for example, the name is *op.xaosw*.

8.1.1 Default XA connection without resource manager

With *openCPIC-path/xa/xa_connection.without_rm*, a dynamic library is supplied which was created without a connection to a resource manager. This library must be used if OpenCPIC os operated without a resource manager.

When OpenCPIC is installed, this library is made available by default as dynamic library *ocpic_xa_connect*. In other words OpenCPIC is by default operated without resource manager.

8.1.2 Default XA connection with test resource manager

With *openCPIC-path/xa/xa_connection.test_rm*, a dynamic library is also supplied which was created with a connection to a test resource manager. When OpenCPIC is installed, this library is supplied by default.

When you want to operate OpenCPIC with this test resource manager, you must replace the *ocpic_xa_connect* file by this library.

8.1.3 Generating the XA connection module

If OpenCPIC is to be coupled with a resource manager, an XA connection module must be generated in the form of a dynamic library with the name *ocpic_xa_connect*. Proceed as follows:

1. If you are using a resource manager other than Oracle, Informix, or MS SQL-Server, adjust the name of the XA switch to *openCPIC-path/xa/xa_info.c*.
2. Use the C compiler to compile the source code under *openCPIC-path/xa/xa_info.c*.

To do this, you must set the preprocessor variable XA_ SWITCH in accordance with your XA resource manager. The possible values are:

XA_SWITCH=0	No resource manager
XA_SWITCH=1	Oracle
XA_SWITCH=2	Informix
XA_SWITCH=3	MS SQL-Server
XA_SWITCH=4	A different resource manager In this case the name of the XA switch must be adjusted to *xa_info.c*.

3. Link this object together with the XA libraries of the resource manager used to form a dynamic library.
4. Make this library available under *openCPIC-path/xa* with the name *ocpic_xa_connect*.

The next time *ocpic_mgr* is started, OpenCPIC will couple with this resource manager.

If several resource managers are installed on your system, you must positively ensure that the OpenCPIC manager uses the same resource manager as the local application programs do. In other words, the program file *ocpic_mgr* and the local application programs must use the same XA connection and must reference the same resource manager.
How to link local application programs is described in chapter “Creating CPI-C application programs”, Abschnitt ”Linking application programs” on page 85.

Dynamic registration (XA calls *ax_reg(), ax_unreg()*) is not supported by OpenCPIC.

8.2 Recovery

The capability to perform a recovery after the occurrence of an error (e.g. program failure or system crash) is an important feature of global transactions. It ensures a high degree of data consistency and operational reliability.

8.2.1 Log records

For the purpose of recovery following the occurrence of an error, the OpenCPIC manager creates log records in the course of a transaction. A log record contains, in addition to a unique transaction Identifier (Transaction ID), information on the involved resource managers, the current state of the transaction and the error which has occurred. The OpenCPIC manager stores the log records as files in the directory *$OCPICDIR/SYNC*. On completion of a transaction, the log records are deleted. If a transaction is interrupted before its completion by a program or system error, the corresponding log record is saved. On restarting (warm start of the OpenCPIC manager), the existing log records inform the OpenCPIC manager about the transactions open at the time the error occurred, and their processing status.

Log record files are binary files. Their format is described in XAP-TP specification [25].

8.2.2 Recovery measures by the OpenCPIC manager

The OpenCPIC manager performs recoveries in the following error situations:

1. Failure of an application program involved in a transaction

   Depending on the point during the transaction at which the error occurred, the OpenCPIC manager either performs a commit or a rollback. If all the involved dialogs are either approaching or in the precommit phase, a rollback is always initiated. If the application program already entered the commit phase with *tx_commit()* before the failure, the decision to commit or rollback depends on the point during the commit phase at which the failure occurred. If the TX library has already signalled the end of the transaction to the local resource manager with t *xa_end()*, a commit is initiated, otherwise a rollback is initiated.
   Following a program failure, the OpenCPIC manager takes the place of the application program communicating with the local resource manager. To prevent the OpenCPIC manager from being blocked for excessively long periods by XA calls required for completing the transaction, these recovery actions are performed in a separate process. For this, the OpenCPIC manager uses the system call *fork()* to create a child process which terminates independently following completion of the transaction.

2. Failure of the system manager, e.g. due to a system crash or on reception of the signal *SIGKILL*

   If the OpenCPIC manager or the entire system fails during a transaction, a warm start should be performed subsequently. For this, invoke *ocpic_start* **without** switch *-c* (see the section “Starting the OpenCPIC manager” on page 54). The OpenCPIC manager first reads the log records out of the files in the directory *$OCPICDIR/SYNC* and uses this information as a basis for implementing recovery measures for all the transactions active on occurrence of the system failure. Whether a transaction is completed with a commit or rollback depends again on the point during the transaction at which the system failed.
   During warm starts too, the OpenCPIC manager uses *fork()* to generate a separate child process for every transaction to be ended.
   Once the recovery measures have been completed after a warm start, the OpenCPIC manager outputs a corresponding message to the log file *$OCPICDIR/PROT/prot.mgr*.

After a recovery has been completed successfully, the log record files in the directory *$OCPICDIR/SYNC* are deleted. However, this does not apply to the log damage records, which indicate possible inconsistencies (heuristic decisions) on termination of the transaction (refer to the following section “Heuristic decisions”).

As on every warm start of the OpenCPIC manager, **all** the log records in the directory *$OCPICDIR/SYNC* are edited, log damage records no longer required should be deleted or moved to another directory from time to time by the system administrator to prevent unnecessarily long restarts.

8.2.3 Heuristic decisions

If inconsistencies arise on the conclusion of a transaction as a result of heuristic decisions (see page 27), these are reported to the involved transaction managers by the OSI-TP protocol element *TP-HEURISTIC-REPORT-indication* or the XAP-TP primitive *TP_LOG_DAMAGE_IND*.

The OpenCPIC manager then outputs a corresponding message (425) to the log file and saves a log damage record in the directory *$OCPICDIR/SYNC*. As the binary log damage record cannot be read directly, the OpenCPIC manager also outputs the most important items of information in a separate ASCII log file.

The application programs are notified of a heuristic decision in the transaction tree by the return value with *tx_commit()* or. *tx_rollback()*. More details on this are provided in the description of the TX interface [26].

Evaluating log damage information from the OpenCPIC manager

Inconsistencies which have arisen must generally be resolved through appropriate measures implemented by the system administrator. The following sources of information are available here:

– The log files of the resource manager. More details on this are provided in your resource manager documentation.

– The log damage records of the OpenCPIC manager. Their contents are represented in legible form in the related ASCII log file.

The names of the binary log record files are structured as follows:

log.trans_id

trans_id is a unique transaction ID comprising several parts.
Every binary log-damage-record is assigned an ASCII log file with the following name:

logasc.trans_id

Related files have the same *trans_id*.

The information in the log damage record can be of considerable help in resolving inconsistencies. Consequently, do not delete these files until you are sure that all resources have been restored to a consistent state.

Format of an ASCII log file

```
ASCII data in a log damage record file %s


XID
     format ID: %d
     gtrid_length: %d
     bqual_length: %d
     data: %s


Local resource manager
     RM ID: %s
     Open string: %s
     Close string: %s


Local node
     AP PID: %d
     AEQ: %d
     APT: %s
```

```
     Node State: %d
     Log Damage State: %d
     Superior: %d
     Subordinates: %d


Superior node
     AEQ: %d
     APT: %s


Subordinate node
     AEQ: %d
     APT: %s
     Log Damage State: %d
```

Meaning of the output fields

Log damage record file
: Name of the related binary log damage record file

XID
: XID of the related transaction. The structure of an XID is defined in the file *tx.h* (also refer to [26]). It contains the following elements:

  format ID
  : Format identifier

  gtrid_length
  : Length of the global transaction identifier in bytes (1-64)
  The global transaction identifier begins with the first byte of the subsequent *data* section.

  bqual_length
  : Length of the branch qualifier in bytes (1-64)
  The branch qualifier starts immediately after the last byte of the global transaction identifier in the subsequent *data* section.

  data
  : Data section consisting of the global transaction identifier and branch identifier of the length specified in each case. The bytes of the data section are output in hexadecimal form.

Local resource manager
: This section contains information on the local resource manager.

  RM ID
  : Internal identifier of the resource manager

  Open String
  : XA-open string of the resource manager

  Close String
  : XA close string of the resource manager

Local node — This section contains information on the local node of the transaction tree.

- AP PID — Process ID of the application program
- AEQ — Application entity qualifier of the local application
- APT — Application process title of the local application
- Node state — This parameter can assume one of the following values:
  - = 0 — AP_TP_NONE
    The transaction was neither in the ready state (phase 1 of the two-phase commit protocol) nor in the commit state (phase 2 of the two-phase commit protocol) when the heuristic decision was reported.
  - = 1 — AP_TP_READY
    The transaction was in the ready state (phase 1 of the two-phase commit protocol) when the heuristic decision was reported.
  - = 2 — AP_TP_COMMIT
    The transaction was in the commit state (phase 2 of the two-phase commit protocol) when the heuristic decision was reported.
- Log damage state
  This parameter can assume one of the following values:
  - = 1 — AP_TP_HEUR_MIX
    A “heuristic mix condition” was indicated in the sub-tree below the local node.
  - = 2 — AP_TP_HEUR_HAZ
    A “heuristic hazard condition” was indicated in the sub-tree below the local node.
- Superior — This indicates whether the local node has a superior.
  - = 0 — The local node does not have a superior. It is thus the root node of the transaction.
  - = 1 — The local node has a superior. It is thus an intermediate or leaf node of the transaction.
- Subordinates — This indicates the number of subordinates of the local node.

Superior node — This parameter only appears if the local node has a superior. It contains information on the superior node.

- AEQ — Application entity qualifier of the superior node
- APT — Application process title of the superior node

Subordinate node — This parameter appears once for each subordinate and contains information on that subordinate.

- AEQ — Application entity qualifier of the subordinate node
- APT — Application process title of the subordinate node
- Log damage state

  This parameter can assume one of the following values:

  - = 0 — AP_TP_NONE
    No heuristic decision was indicated in the sub-tree below the subordinate.
  - = 1 — AP_TP_HEUR_MIX
    A “heuristic mix condition” was indicated in the sub-tree below the subordinate.
  - = 2 — AP_TP_HEUR_HAZ
    A heuristic hazard condition” was indicated in the sub-tree below the subordinate.

9 OpenCPIC program messages

This chapter contains an explanation of all the messages for the OpenCPIC programs and explains what responses to make, where applicable.

Messages of the OpenCPIC manager are normally output to the *prot.mgr* file in the *$OCPICDIR/PROT* directory (or *openCPIC-path/PROT*, if *OCPICDIR* is not set). The OpenCPIC manager only outputs the messages directly on-screen during initialization. The programs *ocpic_gen*, *ocpic_adm*, *ocpic_sta, ocpic_logdump* and *xatmigen* output their messages to *stderr*.

OpenCPIC uses the NLS (Native Language Support) procedure for language-dependent message outputs. Outputs can be in German or English. It is controlled using the environment variable *$LANG*. If *$LANG* is not set or is invalid, the output is in English. The NLS message catalogs are stored in the */opt/lib/nls/msg/De* and */opt/lib/nls/msg/En* directories at startup.

In this manual, the message texts are given in English. Names in uppercase letters which start with a '*&*' (e. g. *&FILENAME*) serve as placemarkers for variable elements in message texts.

In the message descriptions, *$OCPICDIR* always stands for the value of the environment variable *OCPICDIR*, or the default directory *openCPIC-path*, if this variable is not set.

9.1 Messages of the OpenCPIC manager

All messages of the OpenCPIC manager have an ID which specifies the components of the OpenCPIC manager from which the message originates. The IDs have the following meanings:

ID	Message numbers	Component
BD	001 - 299	Boundary (IPC dealer, interface to XAP-TP component)
TPM	300 - 399	TP Manager (Communication Resource Manager)
TM	400 - 499	TM (Transaction Manager)

The messages of the XAP-TP component are output by the BD component, and all begin with message 100. They do have their own message numbers, however, and are therefore described in a separate section (page 171).

9.1.1 Messages of the BD component

```
OpenCPIC [BD] 001 Start of the OpenCPIC manager with PID &MGRPID
```

Meaning
OpenCPIC manager initialization message. It is output immediately after startup and contains the PID *&MGRPID* of the OpenCPIC manager.

```
OpenCPIC [BD] 002 Version: &VERSION
```

Meaning
OpenCPIC manager initialization message. It contains the current version number *&VERSION* of the OpenCPIC manager.

```
OpenCPIC [BD] 003 Trace option(s) = &TROPTS
```

Meaning
OpenCPIC manager initialization message. This message outputs the trace options valid at startup (*&TROPTS*) (see page 54).

```
OpenCPIC [BD] 004 Call syntax: ocpic_start [-tp] [-tm] [-c]
          -tp : with XAP-TP interface trace
          -tm: with manager trace
          -c: cold start
```

Meaning
You called the OpenCPIC manager with invalid switches or parameters.

Response
Find out about the correct call syntax to use in section “Starting the OpenCPIC manager” on page 54.

```
OpenCPIC [BD] 007 No admittance to the directory &DIRNAME
```

Meaning
The OpenCPIC manager cannot change to the working directory *&DIRNAME* (*$OCPICDIR* or *openCPIC-path*, if *OCPICDIR* is not set).

Response
Check the contents of the *OCPICDIR* environment variable and ensure that the directory specified exists. Restart the OpenCPIC manager.

```
OpenCPIC [BD] 008 Shutdown due to lack of resource(s)
```

Meaning
The attempt to create temporary memory has failed. This has caused the OpenCPIC manager to shut down.

Response
Reduce the memory used by other programs or enlarge the main memory of your system. (The memory required by the OpenCPIC manager can be reduced by generating fewer associations with the parameter ASSOCIATIONS in the PARTAPPL statements). Restart the OpenCPIC manager.

```
OpenCPIC [BD] 009 OpenCPIC manager is already running
```

Meaning
OpenCPIC manager has already been started. Only one OpenCPIC manager can run in an environment. If you wish to start more than one OpenCPIC manager in a system, each one must be started in a separate environment and the *OCPICDIR* environment variable set differently for each OpenCPIC manager.

Response
Use *ocpic_sta* to view the current state of the OpenCPIC manager. The *ps* command shows the processes currently active. If you wish to start the OpenCPIC manager in the same environment, first end the running one using *ocpic_adm -e* or *kill -15*. If you want to run another OpenCPIC manager simultaneously, change the work environment.
If this message appears despite the fact that no process with the name *ocpic_mgr* is active in your environment, check in the *$OCPICDIR/PROT* directory if there is a file with the name *pid_mgr*. If there is, delete it and restart OpenCPIC manager.

```
OpenCPIC [BD] 010 The file pid_mgr cannot be created, errno = &ERRNO
```

Meaning
The OpenCPIC manager cannot create one of its system files in the *$OCPICDIR/PROT* directory. The precise error is specified by the contents of the system variable *errno = &ERRNO*.

Response
Depends on the *&ERRNO*. Check the contents of the *OCPICDIR* directory and ensure that the directory specified exists. The restart the OpenCPIC manager.

```
OpenCPIC [BD] 011 Error opening the input pipe, errno = &ERRNO
```

Meaning
The named pipe *$OCPICDIR/.pipes/.pipe_ocpm* cannot be opened. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.
If the input pipe cannot be opened, no communication is possible between the OpenCPIC manager and the application programs, so the OpenCPIC manager shuts down.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
OpenCPIC [BD] 012 Error opening the configuration file, errno = &ERRNO
```

Meaning
The configuration file *$OCPICDIR/conffile* cannot be opened. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. If necessary, generate a new configuration file using *ocpic_gen* (see page 49) and restart the OpenCPIC manager.

```
OpenCPIC [BD] 013 Error reading the configuration file, errno = &ERRNO
```

Meaning
The configuration file *$OCPICDIR/conffile* cannot be read. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. If necessary, generate a new configuration file using *ocpic_gen* (see page 49) and restart the OpenCPIC manager.

```
OpenCPIC [BD] 014 Wrong version in configuration file
```

Meaning
The configuration file generated with *ocpic_gen* (called *conffile*) contains an erroneous version number. It was created with a version of the *ocpic_gen* program which is not compatible with the OpenCPIC manager.

Response
Call *ocpic_gen* (see page 49) and restart the OpenCPIC manager.

```
OpenCPIC [BD] 015 Only &NUMBER of the desired instances could be created.
```

Meaning
During the initialization phase, the OpenCPIC manager attempts to generate an XAP-TP dialogue instance for each association (in addition to a control instance for the monitoring of commitment, rollback and recovery). The number of dialogue instances to be generated

is the sum of all desired associations (parameter ASSOCIATIONS) of all configured partners (PARTAPPL statements). The message says that not all desired instances were prepared by the XAP-TP component. This can indicate an internal error in the XAP-TP component.

Response
Check the contents of the *$OCPICDIR/PROT* directory, especially the log files of the XAP-TP trace, and contact your service.

```
OpenCPIC [BD] 016 Initialization of the OpenCPIC manager finished
```

Meaning
If this message appears on the screen, the OpenCPIC manager has finished its initialization phase and is now ready to take requests from local application programs or remote applications.

```
OpenCPIC [BD] 017 OpenCPIC manager stopped
```

Meaning
This message is written last in the log file when the OpenCPIC manager ends normally (i. e. on the basis of *ocpic_adm -e* or *kill -15*).

```
OpenCPIC [BD] 018 Crash of OpenCPIC manager in module &MNAME , error number = &ERRNUM, error
          code = &ERRCODE
```

Meaning
The OpenCPIC manager crashed due to a serious internal error.

Response
Check the contents of the *$OCPICDIR/PROT* directory and restart the OpenCPIC manager. Contact your service.

```
OpenCPIC [BD] 019 Shutdown by the XAP-TP provider: &NAME
```

Meaning
The OpenCPIC manager crashed due to a serious error in the XAP-TP component.

Response
Check the contents of the *$OCPICDIR/PROT* directory and restart the OpenCPIC manager. Contact your service.

```
OpenCPIC [BD] 020 Error on signal handling; errno = &ERRNO
```

Meaning
An internal error occurred when initializing the signal handling of the OpenCPIC manager. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
OpenCPIC [BD] 025 Overflow of synchronous pip to the AP with PID &APPID
```

Meaning
An overflow of the named pipe occurred during communication between the OpenCPIC manager and a local application program. The OpenCPIC manager retries to write into the pipe after 1 second.

Response
If this message should appear frequently, check if the local AP with the PID specified is still active.

```
OpenCPIC [BD] 030 Dump file &FILENAME cannot be opened
```

Meaning
The OpenCPIC tried to create a dump file *&FILENAME* due to an internal error or an administration command. The file could not be opened. This means that the service does not provide a dump file for error diagnostics.

```
OpenCPIC [BD] 035 Unknown internal message received
OpenCPIC [BD] 040 Internal message of invalid length (&NUMBER byte) received
```

Meaning
These messages indicate an error during communication between the OpenCPIC manager and the (local) administration programs.

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [BD] 045 Read on input pipe failed with error ENOBUFS
```

Meaning
While reading with *read()* from a named pipe, the error *errno = NOBUFS* occurred. This means that there is a momentary memory bottleneck. The OpenCPIC manager repeats the attempt to read.

Response
If this error occurs frequently, or if the OpenCPIC manager shuts down after the maximum number of repetitions (25), contact your system administrator.

```
OpenCPIC [BD] 046 Write into pipe to AP with PID &APPID failed with error ENOBUFS
```

Meaning
While writing with *write()* into a named pipe, the error *errno = ENOBUFS* occurred. This means that there is a momentary memory bottleneck. The OpenCPIC manager repeats the attempt to write.

Response
If this error occurs frequently, or if the OpenCPIC manager shuts down after the maximum number of repetitions (25), contact your system administrator.

```
OpenCPIC [BD] 050 ap_set_env failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 051 ap_get_env failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 052 ap_free failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 053 ap_bind failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 054 ap_close failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 055 ap_init_env failed for instance &INSTNO, mit ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 056 ap_open failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 057 ap_rcv failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 058 ap_snd failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 059 ap_poll failed for instance &INSTNO, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 060 apext_adm (operation code &OPCODE) failed, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 061 apext_att failed, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 062 apext_det failed, ap_errno = &APERRNO: &ERRTXT
OpenCPIC [BD] 063 apext_init for type&INITTYPE , ap_errno = &APERRNO: &ERRTXT
```

Meaning
These messages indicate an internal error when accessing the XAP-IP interface.

&INSTNO	Number of the XAP-IP instance
&APERRNO	XAP-TP error number
&ERRTXT	XAP-TP error text

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [BD] 070 The local AP &APNAME cannot be started: program path not present
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* because of a request to set up a dialogue. The program, which you specified in the generation statement LOCAPRO (parameter PATH), does not exist in your system.

Response
Install the appropriate application program under the specified pathname or rectify your generation file. In the latter case, the configuration file must be regenerated with *ocpic_gen* and the OpenCPIC manager restarted.

```
OpenCPIC [BD] 071 The local AP &APNAME cannot be started: login name &LGNAME not present
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* under the user ID *&LGNAME* because of a request to set up a dialogue. The program, which you specified in the generation statement LOCAPRO (parameter LOGIN), does not exist in your system.

Response
Enter the user ID in your system or rectify the generation file. In the latter case, the configuration file must be regenerated with *ocpic_gen* and the OpenCPIC manager restarted.

```
OpenCPIC [BD] 072 The local AP &APNAME cannot be started:
           fork() system call failed with errno = &ERRNO
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* because of a request to set up a dialogue. A system error occurred during the start of a child process with *fork()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
OpenCPIC [BD] 073 The local AP &APNAME cannot be started: change to HOME directory &DIRNAME
           failed with errno = &ERRNO
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* because of a request to set up a dialogue. In the child process, the attempt to change the directory using *chdir* to the *HOME* directory *&DIRNAME* of the configured user ID (parameter LOGIN in the generation statement LOCAPRO) failed. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. If necessary, modify the access rights.

```
OpenCPIC [BD] 074 The local AP &APNAME cannot be started:
           set group id to &GRPNAME failed with, errno = &ERRNO
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* because of a request to set up a dialogue using *fork()*. In the child process, the attempt to set the user group using *setgid()* according to the configured user ID *&GRPNAME* (parameter LOGIN in the generation statement LOCAPRO) failed. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
OpenCPIC [BD] 075 The local AP &APNAME cannot be started:
           set user id to &USRNAME failed with errno = &ERRNO
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* because of a received request to set up a dialogue. In the child process, the attempt to set the user ID according to the user ID using *setuid()* (parameter LOGIN in the generation statement LOCAPRO) failed. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
OpenCPIC [BD] 076 The local AP &APNAME cannot be started: execve failed with errno = &ERRNO
```

Meaning
The OpenCPIC manager tried to start the locally configured application program *&APNAME* because of a request to set up a dialogue. In the child process, the attempt to start the program using *execve()* (parameter PATH in the generation statement LOCAPRO) failed. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check if the execution rights for your local application program are assigned properly.

```
OpenCPIC [BD] 077 Child process terminated
```

Meaning
Tis message appears after one of the messages 073 - 076. The child process generated using *fork()* is ended due to an error while starting a local application program. The previous message provides information on the cause.

```
OpenCPIC [BD] 100 Message &XAPNUM of the XAP-TP provider: &MSGNUM ...
```

Meaning
This message introduces an internal system message of the XAP-TP component. The text which follows is dependent on the message number *&MSGNUM*. the XAP-TP component messages are listed in a separate section (page 171).

&XAPNUM	internal error message
&MSGNUM	message number

9.1.2 Messages of the TPM component

```
OpenCPIC [TPM] 301 CPI-C request from unknown originator (PID &APPID) received
OpenCPIC [TPM] 302 CPI-C request with unknown conversation ID &CONVID received from
           originator with PID &APPID
```

Meaning
The OpenCPIC manager received a CPI-C request from an unknown application program and/or with an invalid conversation ID. This indicates an internal error.

&APPID	PID of the application program
&CONVID	conversation ID

Response
Check if your application program is linked with the current version of the library *libocpic.a*. Restart the application program. If it happens again, turn on the CPI-C and manager traces. Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TPM] 303 Conversation cannot be built up, maximum association number reached
           (AP PID &APPID, conversation ID &CONVID)
```

Meaning
The maximum permissible number of associations has already been set up and all associations are currently occupied. Since no more associations can be set up, the OpenCPIC manager cannot build up a conversation.
&APPID PID of the application program
&CONVID conversation ID

Response
The maximum number of possible associations is generated in the PARTAPPL statement with the parameter ASSOCIATIONS. If you wish to increase this number, you must modify the configuration file, use *ocpic_gen* to generate a new configuration file and then restart the OpenCPIC manager.

```
OpenCPIC [TPM] 304 Conversation cannot be built up
           Reason: association allocation rejected
           (AP PID &APPID, conversation ID &CONVID)
```

Meaning
The OpenCPIC tried to allocate an association with *return_control = CM_IMMEDIATE* due to an *Allocate (CMALLC)* from a local AP, and did not immediately receive an association from the XAP-TP component. The *CMALLC* is therefore answered with *CM_UNSUCCESSFUL*.
&APPID PID of the application program
&CONVID conversation ID

```
OpenCPIC [TPM] 311 Allocate() of local application program (PID &APPID) rejected
           Partner application: &PARTNAME
           Reason: Partner application not configured
```

Meaning
A local AP tried to set up a conversation with a partner application which is not configured. Either an erroneous *symof thet_name* was entered at *Initialize_Conversation (CMINIT)* or a non-generated partner application as specified with *Set_Partner_LU_Name (CMSPLN)*.
&APPID PID of the local application program
&PARTNAME Name of the desired partner application

Response
Either correct your application program or generate the desired partner application. If you modify your generation, you must generate a new configuration file with *ocpic_gen* as well as shut down and then restart the OpenCPIC manager.

```
OpenCPIC [TPM] 313 Allocate() of local application program (PID &APPID, conversation ID
           &CONVID) rejected
           Reason: Actual transaction is being terminated
```

Meaning

A local AP tried to set up a protected conversation using *Allocate (CMALLC)* while the current transaction was already in the termination phase. At this point, no more conversations/dialogues can be added to the transaction.

&APPID PID of the local application program
&CONVID conversation ID

```
OpenCPIC [TPM] 314 Allocate() of local application program (PID &APPID) rejected
           Partner application: &PARTNAME
           Reason: Invalid security parameters
```

Meaning

A local AP tried to set up a conversation using *conversation_security_type = CM_SECURITY_PROGRAM*, whereby either *security_user_ID* or *security_password* or both were not set. The *CMALLC* therefore returns *CM_PARAMETER_ERROR*.

&APPID PID of the local application program
&PARTNAME Name of the desired partner application

Response

Enter a user ID and a password or modify the value of *conversation_security_type*. The user ID and password are defined either in the generation statement SYMDEST (parameters SEC-UID and SEC-PASS) or with the CPI-C calls S*et_Conversation_Security_Password (CMSCSP)* and S*et_Conversation_Security_User_ID (CMSCSU)*. The *conversation_security_type* can be set either in the generation file (SYMDEST statement, parameter SEC-TYPE) or with the CPI-C call S*et_Converstaion_Security_type(CMSCST)*.

```
OpenCPIC [TPM] 315 Allocate() of local application program (PID &APPID) rejected
           Partner application: &PARTNAME
           Reason: CCR syntax not declared in application context
```

Meaning

A local application program tried to use *Allocate (CMALLC)* to set up a protected conversation with a partner application whose application context does not contain the CCR syntax.

&APPID PID of the local application program
&PARTNAME Name of the desired partner application

Response

If you want to have protected conversations with this partner application, you have to modify the current generation. You can either set another application context which contains the CCR syntax (parameter APPL-CONTEXT in the PARTAPPL statement), or you can explicitly specify that the application context generated by you is to contain the CCR syntax (parameter CCR=YES). If you modify your generation, you have to generate a new configuration file with *ocpic_gen* as well as shut down and then restart the OpenCPIC manager.

```
OpenCPIC [TPM] 320 Passive conversation startup request (CMACCI/CMACCP) rejected (AP PID
           &APPID)
           reason: illegal multiple accept
```

Meaning
A local AP which already is conducting a conversation with a superordinate AP using *sync_level CM_SYNC_POINT* or *CM_SYNC_POINT_NO_CONFIRM* tried to accept another conversation using *Accept_Conversation (CMACCP)* or *Accept_Incoming (CMACCI)*. This is not permitted in OpenCPIC, see also Remark 1 on page 81. The call is returned with value *CM_PRODUCT_SPECIFIC_ERROR*. In the case of *CMACCI*, the value of the *conversation_state* does not change.

&APPID	PID of the application program in question

```
OpenCPIC [TPM] 321 Passive conversation startup request (CMACCI/CMACCP) rejected (AP PID
           &APPID)
           reason: illegal accept at root node
```

Meaning
This message is output when a local application program attempts to accept a conversation using *Accept_Conversation (CMACCP)* or *Accept_Incoming (CMACCI)* and at least one of the following conditions is met:
- the AP already has dialogue to subordinate partners with *sync_level CM_SYNC_POINT* or *CM_SYNC_POINT_NO_CONFIRM*
- the AP has already called *tx_begin()*

This means that the AP of the root nodes of the current or next transaction is rejected. An AP can only be part of one transaction at a time, and transaction trees can only be extended downwards (i.e. by adding more subordinates). See also Remark 1 on page 81. The *CMACCP* or *CMACCI* call is rejected with the return value *CM_PRODUCT_SPECIFIC_ERROR*. In the case of *CMACCI*, the value of the *conversation_state* does not change.

&APPID	PID of the application program in question

```
OpenCPIC [TPM] 350 XAP-TP provider rejects association allocation
           (XAP-TP dialogue &INSTNO, AP PID &APPID, conversation ID &CONVID)
           result = &RESULT, source = &SOURCE, reason = &REASON
```

Meaning
The OpenCPIC manager tried to set up an association due to an *Allocate (CMALLC)* from a local AP. The local XAP-TP component has rejected the association allocate request and the *CMALLC* is returned with the value *CM_ALLOCATE_FAILURE(_RETRY)* or *CM_UNSUCCESSFUL*.

&INSTNO	number of the XAP-TP instance
&APPID	PID of the application program in question
&CONVID	conversation ID
&RESULT	result of the association allocate request (see below, table A)
&SOURCE	source of rejection (see below, table B)
&REASON	reason for rejection (see below, table C)

Table A: Meaning of *result = &RESULT*

result	XAP Code	Meaning
1	AP_REJ_PERM	Association allocation request rejected permanently
2	AP_REJ_TRAN	Association allocation request rejected transiently

Table B: Meaning of *source = &SOURCE*

source	XAP Code	Meaning
0	AP_ACSE_SERV_USER	ACSE service user
1	AP_ACSE_SERV_PROV	ACSE service provider
2	AP_PRES_SERV_PROV	Presentation service provider
7	AP_APM_SERV_PROV	APM service provider (Association Pool Manager)

Table C: Meaning of *reason = &REASON*

reason	XAP/XAP-TP Code	Meaning
-1	AP_TP_NRSN	No reason given
0	AP_TP_CCR_V2_NAVAIL	CCR Version 2 is not available
1	AP_TP_VER_NAVAIL	Version incompatibility
2	AP_TP_CW_REJ	Contention winner property rejected
3	AP_TP_BM_REJ	Bid mandatory value rejected
6	AP_TP_BAD_AET	Local or remote application entity title (AET) invalid
7	AP_TP_BAD_POOL	No association pool found for the AET specified
58	AP_TP_ASSOC_NVAIL	All associations from the association pool are occupied, but the pool limits permit the building of further associations. The XAP-TP component has started building more associations. (only possible for *return_control = CM_IMMEDIATE*).
59	AP_TP_POOL_LIMIT	All associations from the association pool are occupied, and the pool limits do not permit the building of further associations (only possible for *return_control = CM_IMMEDIATE*).
60	AP_TP_POOL_TIMEOUT	The timer for the association pool has run out. This timer determines how long to wait for an association to become free when all associations from the pool are occupied.

reason	XAP/XAP-TP Code	Meaning
61	AP_TP_DIALOGUE_REFUSED	The association is to be occupied with a protected conversation and the current transaction is in the final phase. At this point, no more dialogues can bes added to the transaction tree.

```
OpenCPIC [TPM] 351 XAP-TP provider reports association lost
          (XAP-TP dialogue &INSTNO, AP PID &APPID, conversation ID &CONVID)
          source = &SOURCE, reason = &REASON, event = &EVENT
```

Meaning

The local XAP-TP reported the loss of an association which was occupied with a dialogue.

&INSTNO number of the XAP-TP instance

&APPID PID of the application program in question

&CONVID conversation ID

&SOURCE source of the association loss (see below, table D)
If *&SOURCE* equals 0 or 1, one of the XAP log elements *A-ABORT-IND* or *A-RELEASE-IND* was received. If *&SOURCE* equals 2, the XAP log element *A-PABORT-IND* was received.

&REASON reason for the association loss. The value for *&REASON* depends on *&SOURCE*. If *&SOURCE* equals 0 or 1, one of the XAP log elements *A-ABORT-IND* or *A-RELEASE-IND* was received
If *&SOURCE* equals 2, the XAP log element *A-PABORT-IND* was received. (See below, tables E.1 to E.3).

&EVENT in this version, always equals -1

Table D: Meaning of *source = &SOURCE*

source	XAP Code	Meaning
0	AP_ACSE_SERV_USER	ACSE service user
1	AP_ACSE_SERV_PROV	ACSE service provider
2	AP_PRES_SERV_PROV	Presentation service provider
7	AP_APM_SERV_PROV	APM service provider (Association Pool Manager)

Table E.1: Meaning of *reason = &REASON* for *&SOURCE* equals 0 or 1

reason	XAP-TP Code	Meaning
-1	AP_RSN_NOVAL	*A-RELEASE-IND*: no reason given
0	AP_REL_NORMAL	*A-RELEASE-IND*: normal release request
1	AP_REL_URGENT	*A-RELEASE-IND*: urgent release request
12	AP_TP_ABORTED	*A-ABORT-IND*
30	AP_REL_USER_DEF	*A-RELEASE-IND*: user-defined release request

Table E.2: Meaning of *reason = &REASON* for *&SOURCE* equals 2

reason	XAP Code	Meaning
-1	AP_RSN_NOVAL	No reason given
0	AP_NSPEC	Reason for aborting
1	AP_UNREC_PPDU	Unknown presentation protocol data unit received
2	AP_UNEXPT_PPDU	Unknown presentation protocol data unit expected
3	AP_UNEXPT_SSPRIM	Unexpected session service primitive received
4	AP_UNREC_PPDU_PARM	Presentation protocol data unit with unknown parameters received
5	AP_UNEXPT_PPDU_PARM	Presentation protocol data unit with unexpected parameters received
6	AP_INVAL_PPDU_PARM	Presentation protocol data unit with invalid parameters received

Table E.3: Meaning of *reason = &REASON* for *&SOURCE* equals 7

reason	XAP Code	Meaning
5	AP_TP_IN_DIALOGUE	A dialogue from a partner occupied the association.

```
OpenCPIC [TPM] 352 Begin dialogue request rejected by partner (XAP-TP user)
          (XAP-TP dialogue &INSTNO, AP PID &APPID, conversation ID &CONVID)
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-confirmation* with *Result = rejected* from the communication partner. The rejection of the dialogue is performed by the remote XAP-TP user.

&INSTNO number of the XAP-TP instance
&APPID PID of the application program in question
&CONVID conversation ID

```
OpenCPIC [TPM] 353 Begin dialogue request rejected by partner (XAP-TP provider)
          (XAP-TP dialogue &INSTNO, AP PID &APPID, conversation ID &CONVID)
          reason = &REASON
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-confirmation* with *Result = rejected* from the communication partner. The rejection of the dialogue is performed by the remote XAP-TP user.

&INSTNO number of the XAP-TP instance
&APPID PID of the application program in question
&CONVID conversation ID
&REASON reason for rejection (see below, table F)

Table F: Meaning of *reason = &REASON*

reason	XAP-TP Code	Meaning
-1	AP_TP_NRSN	No reason given
1	AP_TP_TPSUT_UNKNOWN	The specified TPSU title is unknown by the partner.
2	AP_TP_TPSUT_NVAIL_PERM	The specified TPSU title is permanently unavailable at the partner.
3	AP_TP_TPSUT_NVAIL_TRAN	The specified TPSU title is transiently unavailable at the partner.
4	AP_TP_TPSUT_NEEDED	The partner application has received a *TP-BEGIN-DIALOGUE-indication* without a TPSU title.
5	AP_TP_FU_NSUP	One or more of the desired functional units are not supported by the partner.
6	AP_TP_FU_COMB_NSUP	The desired combination of functional units is not supported by the partner.
7	AP_TP_ASSOC_RES	The association which was occupied for the dialogue is already being used by the partner application.
8	AP_TP_RECIPIENT_UNKNOWN	The application entity parameters which were entered in the *TP-BEGIN-DIALOGUE-request* do not identify a known application entity invocation (AEI).

```
OpenCPIC [TPM] 360 Incoming dialogue rejected: shared control requested
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner with which the functional unit *Shared Control* is desired. Since OpenCPIC does not support this functional unit, the dialogue is rejected by the OpenCPIC manager.

```
OpenCPIC [TPM] 361 Incoming dialogue rejected: no local TPSU title indicated
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner without specification of a TPSU title. The dialogue request is therefore rejected by the local OpenCPIC manager.

```
OpenCPIC [TPM] 362 Incoming dialogue rejected: invalid type of local TPSU title
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner with an invalid TPSU title type. The dialogue request is therefore rejected by the local OpenCPIC manager.

Response
The OpenCPIC manager accepts as TSPU title only the types *PRINTABLE_STRING* and *T61_STRING*. Modify the type of the TPSU title in the configuration or in the application programs on the side of the partner. If the partner is an OpenCPIC manager, correct the parameter PARTNER-TYPE in the SYMDEST statement.

```
OpenCPIC [TPM] 363 Incoming dialogue rejected: local AP &APNAME not configured
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner for the TPSU title *&APNAME* which is not known as a local name. The dialogue request is therefore rejected by the local OpenCPIC manager.

Response
If your local application is to accept dialogues for the names *&APNAME*, you must make this known to the OpenCPIC manager, either via generation (LOCAPRO statements) or using the CPI-C call *Specify_Local_TP_Name (CMSLTP)*.

```
OpenCPIC [TPM] 364 Incoming dialogue rejected: local AP &APNAME cannot be started
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner for the TPSU title *&APNAME* which was made known to the OpenCPIC manager via generation
(LOCAPRO statement). The OpenCPIC manager then tried to start the local application and failed. The dialogue request is therefore rejected by the local OpenCPIC manager.

Response
Check the pathname in the LOCAPRO statement of your generation and the execution rights of the file in question.

```
OpenCPIC [TPM] 365 Incoming dialogue rejected: invalid type of remote AP title
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner and could not decode the application process title (APT) of the sender. The dialogue request is therefore rejected by the local OpenCPIC manager. This error should only occur when communicating with foreign systems.

Response
If possible, modify the behavior of the partner application. It may be that it is not possible to communicate with this partner application.

```
OpenCPIC [TPM] 366 Incoming dialogue rejected: invalid type of remote AE qualifier
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner and could not decode the application entity qualifier (AEQ) of the sender. The dialogue request is therefore rejected by the local OpenCPIC manager. This error should only occur when communicating with foreign systems.

Response
If possible, modify the behavior of the partner application. It may be that it is not possible to communicate with this partner application.

```
OpenCPIC [TPM] 367 Incoming dialogue rejected: unknown partner application
```

Meaning
The OpenCPIC manager received from the partner a *TP-BEGIN-DIALOGUE-indication* from a partner application which is not known in the local application. The dialogue request is therefore rejected by the local OpenCPIC manager.

Response
Define the partner application with the PARTAPPL statement in your generation file, generate a new configuration file using *ocpic_gen*, then exit and restart the OpenCPIC manager.

```
OpenCPIC [TPM] 368 Incoming dialogue rejected: cannot decode local TPSU title
```

Meaning
The OpenCPIC manager received a *TP-BEGIN-DIALOGUE-indication* from the partner and could not decode the local TPSU title. The dialogue request is therefore rejected by the local OpenCPIC manager. This error should only occur when communicating with foreign systems.

Response
If possible, modify the behavior of the partner application. It may be that it is not possible to communicate with this partner application.

```
OpenCPIC [TPM] 369 Incoming dialogue rejected: CCR syntax not negotiated for this partner
```

Meaning
The OpenCPIC manager received from the partner a *TP-BEGIN-DIALOGUE-indication* with which the functional unit *Commit* was desired. However, in the local generation, the CCR syntax was not negotiated for this partner application (parameter APPL-CONTEXT or CCR in the PARTAPPL statement). The dialogue request is therefore rejected by the local OpenCPIC manager.

Response
If you want to have dialogues with this partner application which support the functional unit *Commit*, you must make this known to the OpenCPIC manager via generation. Modify the generation file, and exit and then restart the OpenCPIC manager.

```
OpenCPIC [TPM] 370 XAP-TP dialogue &INSTNO aborted by partner (XAP-TP provider)
```

Meaning
The OpenCPIC manager received a *TP-P-ABORT-indication* for a dialogue, i. e. the dialogue was aborted by the remote XAP-TP component
&INSTNO number of the XAP-TP instance

```
OpenCPIC [TPM] 371 XAP-TP dialogue &INSTNO aborted by partner (XAP-TP user)
```

Meaning
The OpenCPIC manager received a *TP-P-ABORT-indication* for a dialogue, i. e. the dialogue was aborted by the remote XAP-TP user
&INSTNO number of the XAP-TP instance

```
OpenCPIC [TPM] 372 Received log data:
```

Meaning
This message is output with messages 370 and 371 if the received log element contained log data. OpenCPIC does not log data, i. e. no log data will be given to the partner if a dialogue is terminated from the local side. When communicating with foreign systems, however, it is possible for log data to be sent by the partner if a dialogue is terminated. Since the CPI-C call *Extract_Log_Data (CMELD)* is not supported, the resulting log data could only be extracted from the file *prot.mgr*.

9.1.3 Messages of the TM component

```
OpenCPIC [TM] 401 TX request from unknown originator (PID &APPID) received (&MODNAME )
```

Meaning
The OpenCPIC manager received a request from an unknown application program. The request is thrown out.
&APPID PID of the sending program
&MODNAME module name

Response
Check if your application program is connected to the current version of the library *libocpic.a*. Restart the application program. If it occurs again, turn on the CPI-C and manager traces. Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TM] 402 A state mismatch occurred when called to process tx_commit():
          The combination of the states of the CPI-C conversation (&CONVSTATE) and its
          associated XAP-TP dialogue (&DIASTATE) is invalid for commitment
          or there is an outstanding operation (OI = &OPINC).
```

Meaning
The OpenCPIC manager received a *tx_commit()* request from a local application program in an invalid state. This indicates an internal error in the TX library or in the OpenCPIC manager.

&CONVSTATE	CPI-C state of the conversation
&DIASTATE	XAP-TP state of the dialogue
&OPINC	incomplete operation (1=TRUE, 0=FALSE)

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TM] 403 A state mismatch occurred when called to process tx_rollback():
          The XAP-TP state (&DIASTATE) of the dialogue on the XAP-TP instance &INSTNO
          is invalid for a TP_ROLLBACK_REQ
          or there is an outstanding operation (OI = &OPINC).
```

Meaning
The OpenCPIC manager received a *tx_rollback()*-request from a local application program in an invalid state. This indicates an internal error in the TX library or in the OpenCPIC manager.

&DIASTATE	XAP-TP state of the dialogue
&INSTNO	number of the XAP-TP instance
&OPINC	incomplete operation (1 = TRUE, 0 = FALSE)

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TM] 404 Cannot locate the TP control block for the primitive &PRIMTYPE
```

Meaning
The OpenCPIC manager received from the XAP-TP component a log element whose recipient is not known to the OpenCPIC manager. The log element is thrown out. This error is logged and then the program continues to run as usual.

&PRIMTYPE	type of log element

Response
If this occurs frequently, turn on the XAP-TP and manager traces, back up the log files and contact your service.

```
OpenCPIC [TM] 405 Invalid parameter Reqtypein tm_send_reply():
           Expected Reqtype= &REQEXP (tp), Reqtypeto send = &REQRECV.
```

Meaning
The OpenCPIC manager tried to respond to a TX request from a local application program with an erroneous request type. This indicates an internal error in the OpenCPIC manager. *&REQEXP* and *&REQRECV* indicate the expected and actual request types in hexadecimal form.

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TM] 406 The following log file contains a damaged entry and is not removed:
           &FILENAME
```

Meaning
This message can occur either during a warmstart of the OpenCPIC manager (restart) or during normal operation.
If it occurs during a warmstart, the OpenCPIC manager attempts to end the interrupted transaction using the log records backed up in the *$OCPICDIR/SYNC* directory and then deletes the log record files, with the exception of the files which contain a log damage record.
If it occurs during normal operation, the OpenCPIC manager deletes the log record files when ending a transaction, i. e. when receiving the *TP-COMMIT-COMPLETE-indication* or *TP-ROLLBACK-COMPLETE-indication* for all instances involved. In this case too, the log damage records received during the transaction are not deleted.
A log damage record shows possible inconsistencies between the resource managers involved in a distributed transaction. As it may contain important information on the operation and last status of the interrupted transaction, it can only be deleted by the system administrator. (see also section “Heuristic decisions” on page 141).

Response
Check if the information in the backed up log record file is needed. Log record files are binary files and therefore cannot be read just like that. Their configuration is described in the XAP-TP specification [25]. The OpenCPIC manager outputs on receipt of a log damage record all user-relevant data from the record to an ASCII file (see message 425). This information can be used by the system administrator in conjunction with any logging information from your resource manager to make the state of your data consistent again.

```
OpenCPIC [TM] 407 No access to directory &DIRNAME during restart, errno = &ERRNO
```

Meaning
The OpenCPIC manager´s attempt to open the *&DIRNAME ($OCPICDIR/sync)* directory using the system call *opendir()* failed. The precise reason is specified in the contents of the system variable *errno = &ERRNO*. After this message is output, the OpenCPIC manager shuts itself down.

Response
Depends on *&ERRNO*. If the specified directory already exists, check its access rights and modify if necessary. Check the access rights of superordinate directories and modify if necessary. Restart the OpenCPIC manager.

```
OpenCPIC [TM] 408 Unable to update a log record for the AP with PID &APPID.
          Primitive type in work is &PRIMTYPE.
```

Meaning
The OpenCPIC manager tried to update an existing log record using a received log element. This is necessary, for example, when the transaction manager in the commit phase receives a report of a heuristic decision and the current log ready record needs to be changed to a log damage record. This message indicates that the log record could not be changed, possibly due to a file error or lack of memory space. In response, the transaction manager initiates a rollback of the transaction.
&APPID PID of the application program in question
&PRIMTYPE type of the XAP-TP log element currently being processed

```
OpenCPIC [TM] 409 Unknown primitive received, sptype = &PRIMTYPE
```

Meaning
The OpenCPIC manager received an unknown log element of the type *&PRIMTYPE* from the XAP-TP component. The log element is thrown out. This error is logged, then the program continues as usual.
&PRIMTYPE type of XAP-TP log element

Response
If this occurs frequently, turn on the XAP-TP and manager traces, back up the log files and contact your service.

```
OpenCPIC [TM] 410 TP_READY_ALL_IND without log record received for AP with PID &APPID
```

Meaning
The transaction manager received the log element *TP-READY-ALL-indication* and no log record was given. This may indicate an internal error of the XAP-TP component. In response, the transaction manager initiates a rollback of the transaction.

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TM] 411 log_req_c contained &NUMBER unexpected elements when processing a
          TP_READY_ALL_IND for the AP with PID &APPID
```

Meaning
This message indicates an internal error in the OpenCPIC manager. This error is logged, then the program continues as usual.

Response
Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
OpenCPIC [TM] 412 TP_READY_ALL_IND received during XAP-TP warmstart
```

Meaning
During warmstart, the OpenCPIC manager received the log element *TP-READY-ALL-indication* for a previously interrupted transaction. This should not happen. The log element is thrown out. This error is logged, then the program continues as usual.

```
OpenCPIC [TM] 413 Cannot access log file &FILENAME during restart; system call &SYSCALL ,
          errno = &ERRNO
```

Meaning
During warmstart, an error occurred while opening or reading the log record file *&FILENAME*. *&SYSCALL* is the name of the system call for which the error occurred (e. g. *open*, *read*). The specific cause of the error is specified by the contents of the system variable *errno = &ERRNO*. The log record file is skipped over. This means it is not possible to restart the transaction in question.

Response
Depends on *&ERRNO*. Check the access rights for the specified file and the superordinate directories. Then perform another warmstart.

```
OpenCPIC [TM] 414 No XAP-TP record in log file &FILENAME
```

Meaning
The file *&FILENAME* from the *$OCPICDIR/sync* directory does not contain a XAP-TP log record, but rather local data on the transaction in question. This means that only local resource managers were involved in the transaction, not partner applications. As a result, the transaction manager only executes local restart measures, without the involvement of the XAP-TP component. This message is only output for information purposes.

```
OpenCPIC [TM] 415 Performing recovery (warmstart) for the transaction represented by
          the log file: &FILENAME
```

Meaning
This message is output at the beginning of a restart for a certain transaction. The transaction manager attempts to end in an orderly manner the transaction which was interrupted by a program or system error. The results of the restart are given in subsequent messages.

```
OpenCPIC [TM] 417 fork() failed during recovery, errno = &ERRNO
```

Meaning
The OpenCPIC manager tried to start child a process for the separate execution of restart measures. A system error occurred when the child process was started with *fork()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
OpenCPIC [TM] 418 Transaction termination with TX_UNCHAINED requires dialogue abortion by
          the transaction manager
          (XAP-TP dialogue &INSTNO, AP PID &APPID, conversation ID %d)
```

Meaning

this message appears when a transaction is ended, if *tx_transaction_control = TX_UNCHAINED* and the CPI-C variable *transaction_control = CM_CHAINED_TRANSACTIONS* for one of the conversations involved. In such a case, an automatic dialogue shutdown is performed by the OpenCPIC manager as described in the X/Open CPI-C specification [24]. The next CPI-C call to the conversation in question is answered with *CM_RESOURCE_FAILURE_RETR*.

&INSTNO number of the XAP-TP instance
&APPID PID of the application program in question
&CONVID conversation ID

Response

This situation can be avoided by making the *tx_transaction_control* (TX) and *transaction_control* (CPI-C) correspond. If you wish to start a new transaction immediately after a transaction has ended, set the values to *TX_CHAINED* (TX) and *CM_CHAINED_TRANSACTIONS* (CPI-C). If you do not want this, set the values to *TX_UNCHAINED* (TX) and *CM_UNCHAINED_TRANSACTIONS* (CPI-C).

```
OpenCPIC [TM] 419 tx_begin of AP with PID &APPID transiently rejected
          Reason: internal rollback caused by transaction control mismatch is not
          completed
```

Meaning

Because of the necessary dialogue shutdown as described in message 418. the OpenCPIC manager is executing an internal rollback. Until the end of this rollback, no new transactions can be started. A *tx_begin()* call in this situation is answered with the return value *TX_ERROR* which, as described in [26], shows a transient error.

&APPID PID of the application program in question

Response

Call the *tx_begin()* at a later time. The instructions given for message 418 are also valid for this situation.

```
OpenCPIC [TM] 425 Log damage record received with XAP-TP primitive &PRIMTYPE
```

Meaning

The OpenCPIC manager received a log damage record. The information from the log damage record is output in readable form to a special ASCII log file (see section “Heuristic decisions” on page 141). After heuristic decisions, this information serves to recreate the consistent state of all the resource managers which were involved in the transaction.

&PRIMTYPE type of XAP-TP log element

9.1.4 Messages of the XAP-TP component

The messages of the XAP-TP component have a separate numbering system. They each start with message 100 of the BD component.

The messages except P100 and P101 are described in the openUTM manual „Messages, Debugging and Diagnostics on Unix and Windows Systems“. You will also find there the description of the corresponding error codes and the inserts of the messages (variable elements with names beginning with a '&').

```
P100    Instance allocation timeout:
        &ACPNT,
        &OSLPAP,
        (&REFINST)
```

Meaning
This message is output when the attempt by an XAP-TP instance for a connection to allocation fails within a set time period.
&ACPNT name of the local ACCESS-POINT
&OSLPAP name of the partner in the local application
&REFINST internal name of the XAP-TP instance

```
P101    CMX Error:
        &ACPNT,
        &OSLPAP
```

Meaning
This message is output when a CMX error occurs. The message P012 is also output.
&ACPNT name of the local ACCESS-POINT
&OSLPAP name of the partner in the local application

9.2 Messages of the ocpic_gen generation program

In the following messages, the placemarkers *&LINENO* and *&INVTXT* always have the following meaning:

&LINENO line number in the generation file where the error occurred.

&INVTXT erroneous text in the generation file.

```
ocpic_gen 00 Message catalog ocpic3.cat not opened: errno = &ERRNO (&ERRNAME)
```

Meaning
The NLS message catalog *ocpic3.cat* could not be opened. It is searched for using the information in the environment variable *$LANG* in the */opt/lib/nls/msg/De* or */opt/lib/nls/msg/En* directory. The cause of error is specified by the contents of the system variable *errno* = *&ERRNO*.

Response
Depends on *&ERRNO*. In some cases, you may have to set up the message catalog again, by reinstalling the product if necessary. You can also change the contents of *$LANG*.

```
ocpic_gen 02 Unknown text in line &LINENO - not read from &INVTXT onward
ocpic_gen 03 No comma permitted at the beginning of the line (error in line &LINENO)
ocpic_gen 04 No comma permitted at the end of a statement (error in line &LINENO)
ocpic_gen 05 Comma is expected in line &LINENO or &LINENO
ocpic_gen 06 Comma at the wrong place in line &LINENO
```

Meaning
These messages indicate syntax errors in the generation file. The correct syntax for the generation file is described in chapter “Generating OpenCPIC” on page 29.

Response
Correct the generation file and restart *ocpic_gen*.

```
ocpic_gen 07 Pathname must begin with / (error in line &LINENO - starting at &INVTXT)
```

Meaning
You entered an erroneous pathname *&INVTXT* in the LOCAPRO statement with the parameter PATH. The pathname must be fully qualified, i. e., it must begin with a slash (/).

```
ocpic_gen 08 Invalid symof thet_name (error in line &LINENO - starting at &INVTXT
```

Meaning
The *symof thet_name* in the SYMDEST statement contains invalid characters. Only characters from the character set 01134 are valid, i. e. uppercase letters and numbers.

```
ocpic_gen 09 Number expected (error in line &LINENO - starting at &INVTXT)
```

Meaning
You entered a non-numeric character (*&INVTXT*) as a value for a numeric parameter.

```
ocpic_gen 10 Invalid locapro_name (error in line &LINENO - starting at &INVTXT)
```

Meaning
The *locapro_name* contains invalid characters. Valid characters include letters, numbers, round brackets, periods (.), plus signs (+), minus signs (-), slashes (/), question marks (?), colons (:), apostrophes (‘) and equal signs (=).

```
ocpic_gen 11 Invalid name (error in line &LINENO - starting at &INVTXT)
```

Meaning
The name contains invalid characters. Valid characters include all printable characters except blanks, semicolons(;), commas (,) and asterisks (*).

```
ocpic_gen 12 PARTAPPL must be in the same line as partappl_name (error in line &LINENO)
ocpic_gen 13 LOCAPPL must be in the same line as locappl_name (error in line &LINENO)
ocpic_gen 14 LOCAPRO must be in the same line as locapro_name (error in line &LINENO)
ocpic_gen 15 SYMDEST must be in the same line as symof thet_name (error in line &LINENO)
ocpic_gen 16 APPLICATION-CONTEXT must be in the same line as application_context_name
          (error in line &LINENO)
```

Meaning
The keyword of a statement must not stand alone in a line. Rather, the following name must also appear on the same line.

```
ocpic_gen 20 LOCAPPL statement may only occur once in the generation (error in line &LINENO)
```

Meaning
The generation file contains more than one LOCAPPL statement. This is not permitted.
&LINENO number of the line in the generation file where the error appeared

```
ocpic_gen 21 Invalid number of elements in the array OBJECT-IDENTIFIER (error in line
          &LINENO)
```

Meaning
A parameter of the object identified type (see page 31) must contain a minimum of 2 and maximum of 10 elements.

```
ocpic_gen 22 Syntax error in line &LINENO
```

Meaning
This error indicates a syntax error in the generation file. It appears when parameters are missing, for example, or belong to another statement, or if there is a comma error. The portion of the statement following this error will not be read.

```
ocpic_gen 24 Name &INVTXT too long (error in line &LINENO)
ocpic_gen 25 Name &INVTXT has to be exactly 8 characters long (error in line &LINENO)
```

Meaning
The name *&INVTXT* has the wrong length.

`ocpic_gen 26 Value &INVTXT is not valid (error in line &LINENO)`

Meaning
The parameter value specified is not within the permitted value range.

`ocpic_gen 27 Value for CONNECT too large (error in the statement before line &LINENO)`

Meaning
The parameter CONNECT=*connect_number* in the PARTAPPL statement must not be larger than the maximum number of associations which you generated with the parameter ASSOCIATIONS in the same statement.

`ocpic_gen 28 Value for CONTWIN too large (error in the statement before line &LINENO)`

Meaning
For the parameter CONTWIN=*(contwin_min contwin_max)* in the PARTAPPL statement, the *contwin_max* must not be larger than the maximum number of associations which you generated with the parameter ASSOCIATIONS in the same statement.

`ocpic_gen 29 Name &INVTXT is not defined in any previous PARTAPPL statement (error in line &LINENO)`

Meaning
You entered a *partappl_name* in a SYMDEST statement which is not generated in any PARTAPPL statement. Please note that the PARTAPPL statement must appear before the SYMDEST statement.

`ocpic_gen 30 Name &INVTXT already allocated (error in line &LINENO)`

Meaning
You already generated the name *&INVTXT* in a previous statement. The names in a generation file must be unique.

`ocpic_gen 31 The AE title built from APT and AEQ is already allocated (error in line &LINENO)`

Meaning
You generated an AE title (combination of APT and AEQ) in a PARTAPPL or LOCAPPL statement which was already assigned is a previous PARTAPPL or LOCAPPL statement. The AE titles serve to identify the applications in the network and must therefore by unique.

`ocpic_gen 32 Internal error: &ERRTXT.`

Meaning
An internal error occurred in the program *ocpic_gen* and no configuration file could be generated.

Response
Back up the generation file for diagnostic purposes and contact your service.

```
ocpic_gen 33 parameter APT is mandatory (error in line &LINENO)
ocpic_gen 34 parameter AEQ is mandatory (error in line &LINENO)
```

Meaning
The mandatory parameter shown is missing in a LOCAPPL or PARTAPPL statement. The parameters APT (application process title) and AEQ (application entity qualifier) Must always be specified when generating local or partner applications. Together, they form the AET (application entity title) and are required in OSI-TP for addressing an application.

```
ocpic_gen 35 parameter PARTNER-APRO does not match PARTNER-TYPE (error in line &LINENO)
```

Meaning
You generated a name in a SYMDEST statement with PARTNER-APRO = *partnerap_name* which contains invalid characters. The valid characters are determined by the value generated with PARTNER-TYPE.

```
ocpic_gen 36 context_name &INVTXT is not defined in any previous APPLICATION-CONTEXT
             statement (error in line &LINENO)
```

Meaning
You entered a *context_name* in a PARTAPPL statement which is not generated in an APPLICATION-CONTEXT statement. Please note that the APPLICATION-CONTEXT statement must appear before the PARTAPPL statement.

```
ocpic_gen 37 parameter CONTWIN: contwin-max is less than contwin-min (error in line &LINENO)
```

Meaning
You entered invalid values for the parameter CONTWIN in a PARTAPPL statement. The value *contwin_max* must be larger than the value *contwin_min* (see page 37).

```
ocpic_gen 38 Name &INVTXT reserved (error in line &LINENO)
```

Meaning
You assigned a key word or a reserved word as a name. Please select another name.

```
ocpic_gen 39 Invalid name &INVTXT (error in line &LINENO)
```

Meaning
You entered an invalid name in a LOCAPPL or PARTAPPL statement. Either the one of the name parts NP1-NP5 is too long or the name *&INVTXT* contains too many name parts (i. e. too many separators ‘.’).

```
ocpic_gen 40 parameter PARTNER-APPL is mandatory (error in statement before line &LINENO)
ocpic_gen 41 parameter PARTNER-APRO is mandatory (error in statement before line &LINENO)
```

Meaning
The parameter PARTNER-APPL or PARTNER-APRO is missing in a SYMDEST statement.

```
ocpic_gen 42 parameter SEC-UID missing (SEC-TYPE = PROGRAM) (error in statement before line
          &LINENO)
ocpic_gen 43 parameter SEC-PASS missing (SEC-TYPE = PROGRAM) (error in statement before
          line&LINENO)
```

Meaning
You generated SEC-TYPE=PROGRAM in a SYMDEST statement. In this case, the parameters SEC-TYPE and SEC-PASS are mandatory.

```
ocpic_gen 44 Warning: parameter SEC-UID/SEC-PASS in statement before line &LINENO serves
          no purpose
```

Meaning
You generated SEC-TYPE=NONE in a SYMDEST statement. In this case, the parameters SEC-TYPE and SEC-PASS serve no purpose. The configuration file is generated anyway.

```
ocpic_gen 50 Usage:
          Generation: ocpic_gen [gen_file] [-f conf_file]
          Reverse       : ocpic_gen -r [conf_file]
```

Meaning
You called *ocpic_gen* with the wrong parameters.

Response
Restart *ocpic_gen* using the right parameters.

```
ocpic_gen 51 File &FILENAME cannot be opened |
          errno = &ERRNO &ERRNAME
```

Meaning
The attempt to open the file *&FILENAME* with the system call *open()* failed. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check and change the access rights if necessary.

```
ocpic_gen 52 The version of ocpic_gen does not match with the file &FILENAME
          (&INVVER instead of &VERSION),
          or the file was not generated with ocpic_gen.
```

Meaning
This message only occurs for the switch *-r*. The version number in the input file *&FILENAME* is either wrong (*&INVVER*) or missing. With *ocpic_gen -r*, you can only read configuration files which were generated with the current version of *ocpic_gen*.

```
ocpic_gen 53 LOCAPPL statement missing
```

Meaning
The generation file does not contain a LOCAPPL statement. A generation file must contain exactly one LOCAPPL statement.

`ocpic_gen 54 PARTAPPL statement missing`

Meaning
The generation file does not contain a PARTAPPL statement. A generation file must contain exactly one PARTAPPL statement.

`ocpic_gen 55 The input file contains no generation statements`

Meaning
The generation file is empty or contains only comment lines.

`ocpic_gen 56 STOP (error)`

Meaning
ocpic_gen terminated due to a serious error. No configuration file could be generated. The previous message provides more detailed information on the cause of the error.

Response
Correct the error and restart *ocpic_gen*.

`ocpic_gen 57 Not enough memory available`

Meaning
ocpic_gen requested memory from the operating system using the system function *malloc()*. It did not receive it.

Response
Reduce the memory used by other programs or enlarge the main memory of your system. Then restart *ocpic_gen*.

`ocpic_gen 58 Error writing to file &FILENAME`

Meaning
An error occurred while writing with *fwrite()* to the file *&FILENAME*.

Response
Check the access rights of the given file.

`ocpic_gen 59 Error reading from file &FILENAME`

Meaning
An error occurred while reading with *fread()* from the file *&FILENAME*.

Response
Check the access rights of the given file.

`ocpic_gen 60 Generation in file &FILENAME`

Meaning
This message is output after a successful generation. *ocpic_gen* generates the configuration file *&FILENAME*.

9.3 Messages of the ocpic_adm administration program

```
ocpic_adm 00 Message catalog ocpic3.cat not opened
```

Meaning
The NLS message catalog *ocpic3.cat* could not be opened. It is searched for using the information in the environment variable *$LANG* in the */opt/lib/nls/msg/De* or */opt/lib/nls/msg/En* directory.

Response
In some cases, you may have to set up the message catalog again, by reinstalling the product if necessary. You can also change the contents of *$LANG*.

```
ocpic_adm 01 Usage:
           ocpic_adm -t ton
           ocpic_adm -t mon
           ocpic_adm -t bon
           ocpic_adm -t tof
           ocpic_adm -t mof
           ocpic_adm -t bof
           ocpic_adm -t tfl
           ocpic_adm -t mfl
           ocpic_adm -t bfl
           ocpic_adm -d <partappl-name>
           ocpic_adm -a <partappl_name>
           ocpic_adm -e
           ocpic_adm -l <trace-file>
```

Meaning
You called *ocpic_adm* with the wrong parameters.

```
ocpic_adm 02 Not enough memory available
```

Meaning
ocpic_adm requested memory from the operating system using the system function *malloc()*. It did not receive it.

Response
Reduce the memory used by other programs or enlarge the main memory of your system. Then restart *ocpic_adm*.

```
ocpic_adm 03 Internal error: &ERRCODE.
```

Meaning
An internal error occurred in the program *ocpic_adm*.

Response
Contact your service.

`ocpic_adm 04 Pipe &PIPENAME cannot be opened: errno &ERRNO (&ERRNAME)`

Meaning
The pipe *&PIPENAME* could not be opened with the system call *open()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

`ocpic_adm 05 Pipe &PIPENAME cannot be created: errno &ERRNO (&ERRNAME)`

Meaning
The pipe *&PIPENAME* could not be created with the system call *creat()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

`ocpic_adm 06 OpenCPIC manager is not started`

Meaning
The OpenCPIC manager is not yet started and therefore the administrator command is useless.

`ocpic_adm 07 Error writing in pipe &PIPENAME: errno &ERRNO (&ERRNAME )`

Meaning
The system call *write()* could not be used to write to the pipe *&PIPENAME*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

`ocpic_adm 08 Error reading pipe &PIPENAME: errno &ERRNO (&ERRNAME)`

Meaning
The system call *read()* could not be used to read from the pipe *&PIPENAME*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

`ocpic_adm 09  partappl_name is not generated`

Meaning
You entered an invalid *partappl_name* when calling *ocpic_adm -d* or *ocpic_adm -a*. The *partappl_name* must be generated in the generation file with a PARTAPPL statement.

Response
Check the *partappl_name* and the current configuration. The command *ocpic_gen -r* can be used to view the configuration file currently being used.

`ocpic_adm 11 XAP-TP trace is already active`

Meaning
You called *ocpic_adm -t ton* or *ocpic_adm -t bon* although the XAP-TP trace was already active. The call has no effect on the XAP-TP trace.

`ocpic_adm 12 Manager trace is already active`

Meaning
You called *ocpic_adm -t mon* or *ocpic_adm -t bon* although the manager trace was already active. The call has no effect on the manager trace.

`ocpic_adm 13 XAP-TP trace was not active`

Meaning
You called *ocpic_adm -t tof* or *ocpic_adm -t bof* although the XAP-TP trace was not active. The call has no effect on the XAP-TP trace.

`ocpic_adm 14 Manager trace was not active`

Meaning
You called *ocpic_adm -t mof* or *ocpic_adm -t bof* although the manager trace was not active. The call has no effect on the manager trace.

`ocpic_adm 15 No free associations exist currently`

Meaning
All associations between your local application and the partner application *partappl_name* are currently occupied. The call from *ocpic_adm -d* therefore has no effect.

Response
The command *ocpic_sta* can be used to obtain more detailed information on the state of the associations.

`ocpic_adm 16 No associations exist currently`

Meaning
No associations currently exist between your local application and the partner application *partappl_name*. The call from *ocpic_adm -a* therefore has no effect.

`ocpic_adm 17 System error in the OpenCPIC manager`

Meaning
A system error occurred in the OpenCPIC manager in conjunction with the administration.

Response
Diagnostic help can be found in the file *$OCPICDIR/PROT/prot.mgr*. Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

`ocpic_adm 18 File &FILENAME cannot be opened: errno &ERRNO (&ERRNAME)`

Meaning
The file *&FILENAME* could not be opened with the system call *open()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

`ocpic_adm 20 Error reading PID from file &FILENAME: errno &ERRNO (&ERRNAME)`

Meaning
An error occurred while reading from the file *&FILENAME* with the system call *read()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

`ocpic_adm 21 Error sending signal: errno &ERRNO (&ERRNAME )`

Meaning
An error occurred while sending a signal via *ocpic_adm*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

`ocpic_adm 23 File &FILENAME cannot be created: errno &ERRNO (&ERRNAME)`

Meaning
An error occurred while creating the file *&FILENAME* with the system call *creat()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

`ocpic_adm 24 File &FILENAME cannot be deleted: errno &ERRNO (&ERRNAME)`

Meaning
An error occurred while deleting the file *&FILENAME* with the system call *unlink()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

`ocpic_adm 27 No flush possible because XAP-TP trace is not active`

Meaning
You called *ocpic_adm -t tfl* or *ocpic_adm -t bfl* although the XAP-TP trace was not active. The call has no effect on the XAP-TP trace.

```
ocpic_adm 28 No flush possible because manager trace is not active
```

Meaning
You called *ocpic_adm -t mfl* or *ocpic_adm -t bfl* although the manager trace was not active. The call has no effect on the manager trace.

```
ocpic_adm 29 OCPICDIR = &PATHNAME is too long
```

Meaning
The contents *&PATHNAME* of the environment variable *OCPICDIR* has exceeded the maximum permissible length of 128 characters. *ocpic_adm* therefore terminates.

```
ocpic_adm 30 partappl_name = &NAME is too long
```

Meaning
You entered an erroneous *partappl_name* (*&NAME*) as argument. The *partappl_name* must not exceed 78 characters.

```
ocpic_adm 34 OpenCPIC manager is in initialization phase
```

Meaning
No administration is possible while the OpenCPIC manager is still in the initialization phase.

Response
Call the administration command later.

9.4 Messages of the program ocpic_sta

`ocpic_sta 00 Message catalog ocpic3.cat not opened`

Meaning
The NLS message catalog *ocpic3.cat* could not be opened. It is searched for using the information in the environment variable *$LANG* in the */opt/lib/nls/msg/De* or */opt/lib/nls/msg/En* directory.

Response
In some cases, you may have to set up the message catalog again, by reinstalling the product if necessary. You can also change the contents of *$LANG*.

`ocpic_sta 01 Usage: ocpic_sta [-l]`

Meaning
You called *ocpic_sta* with the wrong parameters.

`ocpic_sta 02 Not enough memory available`

Meaning
ocpic_sta r requested memory from the operating system using the system function *malloc()*. It did not receive it.

Response
Reduce the memory used by other programs or enlarge the main memory of your system. Then restart *ocpic_sta*.

`ocpic_sta 03 Internal error: &ERRCODE.`

Meaning
An internal error occurred in the program *ocpic_sta*.

Response
Contact your service.

`ocpic_sta 04  Pipe &PIPENAME cannot be opened: errno &ERRNO (&ERRNAME)`

Meaning
The pipe specified could not be opened with the system call *open()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

`ocpic_sta 05 Pipe &PIPENAME cannot be created: errno &ERRNO (&ERRNAME)`

Meaning
The pipe *&PIPENAME* could not be created with the system call *creat()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
ocpic_sta 06 Error writing into pipe &PIPENAME: errno &ERRNO (&ERRNAME)
```

Meaning
The pipe *&PIPENAME* could not be written to with the system call *write()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
ocpic_sta 07 Error reading pipe &PIPENAME : errno &ERRNO (&ERRNAME )
```

Meaning
The pipe *&PIPENAME* could not be read from with the system call *read()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Contact your system administrator if necessary.

```
ocpic_sta 08 System error in OpenCPIC manager
```

Meaning
A system error occurred in the OpenCPIC manager in conjunction with the administration.

Response
Diagnostic help can be found in the file *$OCPICDIR/PROT/prot.mgr*. Check the contents of the *$OCPICDIR/PROT* directory and contact your service.

```
ocpic_sta 09 File &FILENAME cannot be opened: errno &ERRNO (&ERRNAME)
```

Meaning
The file *&FILENAME* could not be opened with the system call *open()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

```
ocpic_sta 10 Error reading file &FILENAME: errno &ERRNO (&ERRNAME)
```

Meaning
An error occurred while reading from the file *&FILENAME* with the system call *read()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*.

```
ocpic_sta 11 OCPICDIR = &PATHNAME is too long
```

Meaning
The contents of *&PATHNAME* of the environment variable *OCPICDIR* has exceeded the maximum permissible length of 128 characters. *ocpic_sta* therefore terminates.

`ocpic_sta 12 File &FILENAME cannot be closed: errno &ERRNO (&ERRNAME)`

Meaning
An error occurred while closing the file *&FILENAME* with the system call *close()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*.

`ocpic_sta 13 File &FILENAME cannot be deleted: errno &ERRNO (&ERRNAME )`

Meaning
An error occurred while deleting the file *&FILENAME* with the system call *unlink()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*.

`ocpic_sta 14 Error reading PID from file &FILENAME: errno &ERRNO (&ERRNAME)`

Meaning
An error occurred while reading from the file *&FILENAME* with the system call *read()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*.

`ocpic_sta 15 Error sending signal to OpenCPIC manager: errno &ERRNO (&ERRNAME)`

Meaning
An error occurred while sending a signal via *ocpic_adm*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*.

9.5 Messages of the program ocpic_logdump

The following messages are output by the program *ocpic_logdump*. *ocpic_logdump* is an internal program for the evaluation of trace information and is called by *ocpic_adm -l* and *ocpic_tredit*, among others.

```
ocpic_logdump 11 Error opening &FILENAME, errno = &ERRNO
```

Meaning
The file *&FILENAME* could not be opened with the system call *open()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

```
ocpic_logdump 12 Error reading from &FILENAME, errno = &ERRNO
```

Meaning
An error occurred while reading from the file *&FILENAME* with the system call *open()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*. Check the access rights and modify them if necessary.

```
ocpic_logdump 13 Not enough memory available
```

Meaning
ocpic_logdump requested memory from the operating system using the system function *malloc()*. It did not receive it.

Response
Reduce the memory used by other programs or enlarge the main memory of your system. Then restart the command.

```
ocpic_logdump 14 Error closing &FILENAME, errno = &ERRNO
```

Meaning
An error occurred when closing the file *&FILENAME* with the system call *close()*. The precise reason is specified in the contents of the system variable *errno = &ERRNO*.

Response
Depends on *&ERRNO*.

9.6 Messages of the program xatmigen

The messages of *xatmigen* are in the from `XGnn messagetext...` and are output after *stdout*.

`XG01` `Generation of the Local Configuration Files: &LCF / &DEF / &CODE`

Meaning
Start message of the program .
&LCF name of the generated local configuration files
&DEF name of the generated generation fragments
&CODE string code for character arrays

`XG02` `Generation successfully ended`

Meaning
The LCF was generated, the generation was successfully ended.

`XG03` `Generation successfully ended with warning`

Meaning
The LCF was generated. However, at least one warning was output (e. g. because files were entered which were not required). This warning has no effect on the generation.

`XG04` `Generation ended due to error`
`No file generated.`

Meaning
The LCF was not generated, the generation could not be carried out. The cause can be determined from the previous message.

`XG05` `&FTYPE file '&FNAME'`

Meaning
This message outputs the file currently being processed in the following form:
&FTYPE “description” file contains datastructures
“definition” file contains the LCF input
“LC file” contains the local configuration
&FNAME file name

`XG10` `Call: &PARAM`

Meaning
Syntax error while calling *xatmigen*:
&PARAM call parameters and switches

XG11 `[Error] &FTYPE file 'FNAME' cannot be generated: &REASON`

Meaning
The file *&FNAME* of the types *&FTYPE* cannot be generated.
&REASON contains a detailed reason
&FTYPE GEN = generation fragment file
LC = local configuration file

XG12 `[Warning] File not found.`

Meaning
The definition file or a description file could not be found; it may not exist.

XG13 `[Warning] Too many &OBJECTS, Maximum: &MAXNUM`

Meaning
Message about to many objects found
&OBJECTS subtypes
&MAXNUM maximum number

XG14 `[Error] line &LINE: Syntax error, &HELPTEXT`

Meaning
Syntax error in line *&LINE* in the LC definition file
&HELPTEXT help text

XG15 `[Error] line &LINE: No record definition found for buffer &BUFF`

Meaning
No record definition could be found for the buffer *&BUFF* in line *&LINE*.

XG16 `[Error] line &LINE: Base type error in buffer &BUFF`

Meaning
The syntax description of the buffer *&BUFF* in line *&LINE* of the LCF contains an erroneous base type (int, short etc.) or an invalid string code was entered by COBOL (e. g. "P").

XG17 `[Error] &FTYPE file '&FNAME' cannot be opened. &REASON`

Meaning
The file *&FNAME* of the type *&FTYPE* cannot be opened.
&REASON contains a detailed reason
&FTYPE DEF = LC definition file

XG18 `[Error] &REASON`

Meaning
General error message. *&REASON* contains a detailed description of the error.

XG19 `[Message] New buffer generated: &BUFF`

Meaning
The new buffer *&BUFF* was generated.

XG20 `[Message] Service name ‘&SVC‘shortened to 16 characters!`

Meaning
&SVC: service name

XG21 `[Message] line &LINE: unknown statement line ‘&HELPTEXT‘`

Meaning
Message for the line *&LINE* in the LC definition file
&HELPTEXT help text (part of the LC line)

XG22 `[Message] line &LINE: default value set MODE=‘&TEXT‘`

Meaning
Message for the line *&LINE* in the LC definition file
&TEXT set default service mode

10 Appendix

10.1 Character sets

In OSI-TP, the ASN.1 character string types *PrintableString* and *T61String* are used (see [29]). CPI-C uses the character sets 01134 and 00640.

The character set *T61String* contains all the characters defined in the following code table. The complete CCITT documentation can be found in [34].

Code table in the T6 character set1

	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					

Meaning of abbreviations:

BS	BACKSPACE
CR	CARRIAGE RETURN
CSI	CONTROL SEQUENCE INTRODUCER
ESC	ESCAPE
FF	FORM FEED
LF	LINE FEED
LS0	LOCKING SHIFT ZERO
LS1	LOCKING SHIFT ONE
PLD	PARTIAL LINE DOWN
PLU	PARTIAL LINE UP
SP	SPACE
SS2	SINGLE SHIFT TWO
SS3	SINGLE SHIFT THREE
SUB	SUBSTITUTE CHARACTER

The character sets *PrintableString*, 01134 and 00640 are subsets of the *T61String* character set. The following table shows which characters are valid in each character set.

Table of the *PrintableString*, 01134 and 00640 character sets

Character	PrintableString	01134	00640
SP	x		x
.	x		x
<			x
(	x		x
+	x		x
&			x
*			x
)	x		x
;			x
-	x		x
/	x		x
,	x		x
%			x
_			x
>			x
?	x		x
:	x		x
‘	x		x
=	x		x
“			x
a-z	x		x
A-Z	x	x	x
0-9	x	x	x

10.2 Code conversion tables

The following conversion tables are used for the conversion from ASCII to EBCDIC using the CPI-C call *Convert_Outgoing (CMCNVO)* and from EBCDIC to ASCII using the CPI-C call *Convert_Incoming (CMCNVI)*.

Conversion from ASCII to EBCDIC

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

Conversion from EBCDIC to ASCII

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

Glossary

A term in *italic* font means that it is explained somewhere else in the glossary.
You will find UTM-specific terms in Glossaries of the UTM manuals.

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 point (OSI)

See *service access point*.

ACID properties

Acronym for the fundamental properties of *transactions*: atomicity, consistency, isolation and durability.

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 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 service element (OSI)

An application service element (ASE) represents a functional group of the application layer (layer 7) of the *OSI reference model*.

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*.

CCR (Commitment, Concurrency and Recovery)

CCR is an Application Service Element (ASE) defined by OSI used for OSI TP communication which contains the protocol elements (services) related to the beginning and end (commit or rollback) of a *transaction*. CCR supports the two-phase commitment.

client

Clients of a *UTM application* can be:
- terminals
- UPIC client programs
- transport system applications (e.g. DCAM, PDN, CMX, socket applications or UTM applications which have been generated as *transport system applications*).

Clients are connected to the UTM application via LTERM partners.
openUTM clients which use the OpenCPIC carrier system are treated just like *OSI TP partners*.

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.

contention loser

Every connection between two partners is managed by one of the partners. The partner that manages the connection is known as the *contention winner*. The other partner is the contention loser.

contention winner

A connection's contention winner is responsible for managing the connection. Jobs can be started by the contention winner or by the *contention loser*. If a conflict occurs, i.e. if both partners in the communication want to start a job at the same time, then the job stemming from the contention winner uses the connection.

conversation

In CPI-C, communication between two CPI-C application programs is referred to as a conversation. The communication partners in a conversation are referred to as the *initiator* and the *acceptor*.

conversation ID

CPI-C assigns a local conversation ID to each *conversation*, i.e. the *initiator* and *acceptor* each have their own conversation ID. The conversation ID uniquely assigns each CPI-C call in a program to a conversation.

CPI-C

CPI-C (Common Programming Interface for Communication) is a program interface for program-to-program communication in open networks standardized by X/Open and CIW (**C**PI-C **I**mplementor's **W**orkshop).
The CPI-C implemented in openUTM complies with X/Open's CPI-C V2.0 CAE Specification. The interface is available in COBOL and C. In openUTM, CPI-C can communicate via the OSI TP, *LU6.1* and UPIC protocols and with openUTM-LU62.

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.*

Functional Unit (FU)

A subset of the *OSI-TP* protocol providing a particular functionality. The OSI-TP protocol is divided into the following functional units:

– Dialog
– Shared Control
– Polarized Control
– Handshake
– Commit
– Chained Transactions
– Unchained Transactions
– Recovery

Manufacturers implementing OSI-TP need not include all functional units, but can concentrate on a subset instead. Communications between applications of two different OSI-TP implementations is only possible if the included functional units are compatible with each other.

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”.

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.

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 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*.

LPAP bundle

LPAP bundles allow messages to be distributed to LPAP partners across several partner applications. If a UTM application has to exchange a very large number of messages with a partner application then load distribution may be improved by starting multiple instances of the partner application and distributing the messages across the individual instances. In an LPAP bundle, *open*UTM

is responsible for distributing the messages to the partner application instances. An LPAP bundle consists of a master LPAP and multiple slave LPAPs. The slave LPAPs are assigned to the master LPAP on UTM generation. LPAP bundles exist for both the OSI TP protocol and the LU6.1 protocol.

network selector

The network selector identifies a service access point to the network layer of the *OSI reference model* in the local system.

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.

OpenCPIC

Carrier system for UTM clients that use the *OSI TP* protocol.

OpenCPIC client

OSI TP partner application with the *OpenCPIC* carrier system.

openUTM application

See *UTM application*.

OSI-LPAP bundle

LPAP bundle for *OSI TP* partner applications.

OSI-LPAP partner

OSI-LPAP partners are the addresses of the *OSI TP partners* generated in openUTM. In the case of *distributed processing* via the *OSI TP* protocol, an OSI-LPAP partner for each partner application must be configured in the local application. The OSI-LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned OSI-LPAP partner and not by the application name or address.

OSI reference model

The OSI reference model provides a framework for standardizing communications in open systems. ISO, the International Organization for Standardization, described this model in the ISO IS7498 standard. The OSI reference model divides the necessary functions for system communication into seven logical layers. These layers have clearly defined interfaces to the neighboring layers.

OSI TP

Communication protocol for distributed transaction processing defined by ISO. OSI TP stands for Open System Interconnection Transaction Processing.

OSI TP partner

Partner of the UTM application that communicates with the UTM application via the OSI TP protocol.
Examples of such partners are:

– a UTM application that communicates via OSI TP
– an application in the IBM environment (e.g. CICS) that is connected via openUTM-LU62
– an application of the OpenCPIC carrier system of the openUTM client
– applications from other TP monitors that support OSI TP

outbound conversation (CPI-C)

See *outgoing conversation*.

outgoing conversation (CPI-C)

A conversation in which the local CPI-C program is the *initiator* is referred to as an outgoing conversation. In the X/Open specification, the term “outbound conversation” is used synonymously with “outgoing conversation”.

presentation selector

The presentation selector identifies a service access point to the presentation layer of the *OSI reference model* in the local system.

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.

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 *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.

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.

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 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 TAC (KDCS)

Transaction code used to start a *service*.

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*.

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.

TAC

See *transaction code*.

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.

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 connection

In the *OSI reference model,* this is a connection between two entities of layer 4 (transport layer).

transport selector

The transport selector identifies a service access point to the transport layer of the *OSI reference model* in the local system.

typed buffer (XATMI)

Buffer for exchanging typed and structured data between communication partners. Typed buffers ensure that the structure of the exchanged data is known to both partners implicitly.

UPIC

Carrier system for openUTM clients. UPIC stands for Universal Programming Interface for Communication.

UPIC client

The designation for openUTM clients with the UPIC carrier system.

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 generation

Static configuration of a *UTM application* using the UTM tool KDCDEF and creation of an application program.

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.

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
AP	Application Program
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder - Loader - Starter (BS2000 systems)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Coded Character Set
CCSN	Coded Character Set Name
CICS	Customer Information Control System
CID	Control Identification
CMX	Communication Manager in Unix, Linux and Windows Systems
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000 systems)
DB	Database
DC	Data Communication

DCAM	Data Communication Access Method
DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000 systems)
DMS	Data Management System
DNS	Domain Name Service
DP	Distribted Processing
DSS	Terminal (Datensichtstation)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans™
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GSSB	Global Secondary Storage Area
HIPLEX®	Highly Integrated System Complex (BS2000 systems)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IFG	Interactive Format Generator
ILCS	Inter-Language Communication Services (BS2000 systems)
IMS	Information Management System (IBM)
IPC	Inter-Process Communication
IRV	International Reference Version
ISO	International Organization for Standardization
Java EE	Java Platform, Enterprise Edition
JCA	Java EE Connector Architecture
JDK	Java Development Kit
KAA	KDCS Application Area
KB	Communication Area
KBPRG	KB Program Area
KDCADMI	KDC Administration Interface

KDCS	Compatible Data Communication Interface
KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000 systems)
LSSB	Local Secondary Storage Area
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000 systems)
NB	Message Area
NEA	Network Architecture for BS2000 Systems
NFS	Network File System/Service
NLS	Native Language Support
OLTP	Online Transaction Processing
OML	Object Module Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Process Identification
PIN	Personal Identification Number
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Computer Center Accounting Procedure
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption algorithm according to Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000 systems)
RTS	Runtime System
SAT	Security Audit Trail (BS2000 systems)
SECOS	Security Control System
SEM	SE Manager
SGML	Standard Generalized Markup Language

SLU	Secondary Logical Unit
SM2	Software Monitor 2
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primary Working Area
SQL	Structured Query Language
SSB	Secondary Storage Area
SSO	Single Sign-On
TAC	Transaction Code
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-Specific Long-Term Storage
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaction Mode)
TPR	Privileged Function State in BS2000 systems (Task Privileged)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Non-Privileged Function State in BS2000 systems (Task User)
TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universal Database System
UDT	Unstructured Data Transfer
ULS	User-Specific Long-Term Storage
UPIC	Universal Programming Interface for Communication
USP	UTM Socket Protocol
UTM	Universal Transaction Monitor
UTM-D	UTM Variant for Distributed Processing in BS2000 systems
UTM-F	UTM Fast Variant
UTM-S	UTM Secure Variant

UTM-XML	openUTM XML Interface
VGID	Service ID
VTSU	Virtual Terminal Support
WAN	Wide Area Network
WS4UTM	Web-Services for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (X/Open interface for acess to the resource manager)
XAP	X/OPEN ACSE/Presentation programming interface
XAP-TP	X/OPEN ACSE/Presentation programming interface Transaction Processing extension
XATMI	X/Open Application Transaction Manager Interface
XCS	Cross Coupled System
XHCS	eXtended Host Code Support
XML	eXtensible Markup Language

Related publications

openUTM documentation

[1] **openUTM**
Concepts and Functions
User Guide

[2] **openUTM**
Programming Applications with KDCS for COBOL, C and C++
Core Manual

[3] **openUTM**
Generating Applications
User Guide

[4] **openUTM**
Using openUTM Applications on BS2000 Systems
User Guide

[5] **openUTM**
Using openUTM Applications on Unix, Linux and Windows Systems
User Guide

[6] **openUTM**
Administering Applications
User Guide

[7] **openUTM**
Messages, Debugging and Diagnostics on BS2000 Systems
User Guide

[8] **openUTM**
Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems
User Guide

[9] **openUTM**
Creating Applications with X/Open Interfaces
User Guide

[10] **openUTM**
XML for openUTM

[11] **openUTM Client** (Unix systems)
for the OpenCPIC Carrier System
Client-Server Communication with openUTM
User Guide

[12] **openUTM Client**
for the UPIC Carrier System
Client-Server Communication with openUTM
User Guide

[13] **openUTM WinAdmin**
Graphical Administration Workstation for openUTM
Description and online help system

[14] **openUTM WebAdmin**
Web Interface for Administering openUTM
Description and online help system

[15] **openUTM, openUTM-LU62**
Distributed Transaction Processing
between openUTM and CICS, IMS and LU6.2 Applications
User Guide

[16] **openUTM** (BS2000)
Programming Applications with KDCS for Assembler
Supplement to Core Manual

[17] **openUTM** (BS2000)
Programming Applications with KDCS for Fortran
Supplement to Core Manual

[18] **openUTM** (BS2000)
Programming Applications with KDCS for Pascal-XT
Supplement to Core Manual

[19] **openUTM** (BS2000)
Programming Applications with KDCS for PL/I
Supplement to Core Manual

[20] **WS4UTM** (Unix systems and Windows systems**)**
WebServices for openUTM

[21] **openUTM**
Master Index

Documentation for the Unix, Linux and Windows system environment

[22] **CMX** V6.0 (Unix systems)
Betrieb und Administration (only available in German)
User Guide

[23] **CMX** V6.0
Programming CMX Applications
Programming Guide

X/Open publications

[24] X/Open CAE Specification
Distributed Transaction Processing:
The CPI-C Specification, Version 2
ISBN: 1-85912-135-7

[25] X/Open CAE Specification
ACSE/Presentation:
Transaction Processing API (XAP-TP)
ISBN: 1-85912-091-1

[26] X/Open CAE Specification:
Distributed Transaction Processing:
The TX (Transaction Demarcation) Specification
ISBN:1-85912-094-6

[27] X/Open CAE Specification
Distributed Transaction Processing:
The XA Specification
ISBN: 1-872630-24-3

[28] X/Open CAE Specification
Distributed Transaction Processing
The XATMI Specification
ISBN: 1-85912-130-6

ISO publications

[29] ISO 8824
Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)

[30] ISO 8825
Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

[31] ISO 9804
Information Technology - Open Systems Interconnection - Service Definition for the Commitment, Concurrency and Recovery service element

[32] ISO/IEC 10026
Information Technology - Open Systems Interconnection - Distributed Transaction Processing
Part 1: 1992, OSI TP Model
Part 2: 1992, OSI TP Service
Part 3: 1992, Protocol Specification
Part 4: 1995, Protocol Implementation Conformance Statement (PICS) proforma
Part 5: DIS 1993, Application context proforma and guidelines when using OSI TP
Part 6: 1994, Unstructured Data Transfer (UDT)

Other publications

[33] **NetView/PC Application Programming, Version 1.2**
IBM, SC31-6004

[34] **T.61**
CCITT Recommendation T.61: 1988
Character Repertoire and Coded Character Sets for the International Teletx Service

Index

D