openUTM-Client  V6.0
for the UPIC Carrier System

Client-Server Communication with openUTM
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 9001:2000

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 9001:2000.

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

Copyright and Trademarks

Copyright © Fujitsu Technology Solutions GmbH 2010.
All rights reserved.
Delivery subject to availability; right of technical modifications reserved.

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

1 Preface ................................................................. 11

1.1 Brief description of the openUTM-Client product ............... 13

1.2 Summary of contents and target group .......................... 14

1.3 Summary of contents of the openUTM documentation .......... 15

1.3.1 openUTM documentation ...................................... 15

1.3.2 Documentation for the openSEAS product environment .... 19

1.3.3 README files .................................................. 20

1.4 Changes since the last version of this manual .................. 21

1.5 Notational conventions .......................................... 22

2 Application area ..................................................... 25

2.1 The concept of openUTM-Client ................................ 26

2.2 Client/server communication with openUTM ..................... 29

2.3 UPIC local, UPIC remote and multithreading ................... 30

2.4 Support for UTM cluster applications .......................... 33

3 ActiveX control UpicB.ocx .......................................... 35

3.1 Interfaces of UpicB.ocx .......................................... 40

3.1.1 Properties of UpicB.ocx ...................................... 41

3.1.2 Methods of UpicB.ocx ........................................ 48

3.2 DataReceived event ............................................... 61

3.3 Property Pages window ........................................... 62

3.4 Application example .............................................. 64

3.5 The ActiveX automation server UpicAutSvr.exe ................. 68
## 4 C++ class CUpic

4.1 Introduction ............................................. 69
4.1.1 Configuration using Helper classes CUpicLocAddr and CUpicRemAddr ............. 69
4.1.2 Configuration using a side information file (upicfile) .......................... 70
4.1.3 The CUpic class on thread-capable systems ........................................ 70
4.2 Helper classes ............................................... 71
4.2.1 CUpicLocAddr ........................................... 71
4.2.1.1 Constructors ........................................ 71
4.2.1.2 Member functions .................................... 72
4.2.2 CUpicRemAddr ........................................... 73
4.2.2.1 Constructors ........................................ 73
4.2.2.2 Member functions .................................... 74
4.2.3 CUpic security .......................................... 75
4.3 ClassCUpic .................................................. 76
4.3.1 Constructors ............................................. 76
4.3.2 Property handlers ....................................... 76
4.3.3 Function calls ........................................... 78
4.3.4 Public diagnostic function ................................ 82
4.4 Example ....................................................... 83

## 5 CPI-C interface

5.1 CPI-C terms .................................................. 86
5.2 General structure of a CPI-C application ............................................. 91
5.3 Exchange of messages with a UTM service ........................................... 92
5.3.1 Sending a message and starting a UTM service .................................. 93
5.3.2 Receiving a message, blocking and non-blocking receive ....................... 95
5.3.3 Sending and receiving formats ................................................. 98
5.3.4 UTM function keys .......................................... 102
5.3.5 Cursor position ........................................... 104
5.3.6 Code conversion ............................................ 104
5.3.7 User-defined code conversion for Windows systems ................................ 106
5.4 Communicating with the UTM server .............................................. 107
5.4.1 Communicating in a single-step UTM service .................................... 108
5.4.2 Communicating in a multi-step UTM service ..................................... 110
5.4.3 Communicating in a multi-step UTM service with distributed transaction processing 111
5.4.4 Querying the transaction state .......................................... 112
Contents

Set_Conversation_Security_Type - Setting the security type ........................................... 230
Set_Conversation_Security_User_ID - Setting the UTM user ID ........................................ 232
Set_Conversion -
  Setting the CHARACTER_CONVERSION conversation characteristic .......................... 234
Set_Deallocate_Type - Setting deallocate_type ............................................................... 236
Set_Function_Key - Setting a UTM function key ............................................................... 238
Set_Partner_Host_Name - Setting the partner host name ................................................. 241
Set_Partner_IP_Address - Setting the IP address of the partner application ..................... 243
Set_Partner_LU_Name - Setting the conversation characteristics partner_LU_name ............ 246
Set_Partner_Port - Setting the TCP/IP port for the partner application .......................... 249
Set_Partner_Tsel - Setting the T-SEL of the partner application ..................................... 251
Set_Partner_Tsel_Format - Setting the T-SEL format of the partner application ............... 253
Set_Receive_Timer - Setting the timer for a blocking receive ........................................ 255
Set_Receive_Type - Setting the receive type ..................................................................... 258
Set_Sync_Level - Setting a synchronization level ............................................................. 261
Set_TP_Name - Setting TP-name ....................................................................................... 263
Specify_Local_Host_Name - Setting the local host name .................................................... 265
Specify_Local_Tsel - Setting the T-SEL of the local application ...................................... 267
Specify_Local_Tsel_Format - Setting the TSEL format of the local application .............. 269
Specify_Secondary_Return_Code -
  Setting the properties of the secondary return code ...................................................... 271
5.10 COBOL interface ......................................................................................................... 273

6 XATMI interface ............................................................................................................. 275

6.1 Linking client/server applications .................................................................................. 276
6.1.1 Default server ........................................................................................................... 277
6.1.2 Restart ..................................................................................................................... 277
6.2 Communication paradigms ............................................................................................ 278
6.3 Typed buffers .............................................................................................................. 281
6.4 Program interface ...................................................................................................... 284
6.4.1 XATMI functions for clients .................................................................................... 284
6.4.2 Calls for connecting to the carrier system ............................................................... 285
tpinit - Initializing the client ............................................................................................. 286
tpterm - Signing the client off .......................................................................................... 288
6.4.3 Transaction control ................................................................................................. 289
6.4.4 Mixed operation ...................................................................................................... 289
6.4.5 Administration interface .......................................................................................... 290
6.4.6 Header files and COPY elements ............................................................................ 290
6.4.7 Events and error handling ....................................................................................... 291
### Contents

6.4.8 Creating typed buffers ................................................. 292
6.4.9 Characteristics of XATMI in UPIC .................................... 294

6.5 Configuring ............................................................. 295
6.5.1 Creating the local configuration file ............................... 295
6.5.2 xatmigen or xtgen32 tool ............................................. 300
6.5.3 Configuring the carrier system and UTM partners .................. 303
6.5.3.1 Configuring UPIC ................................................. 303
6.5.3.2 Initialization parameters and UTM generation ................. 304

6.6 Running XATMI applications ........................................... 307
6.6.1 Linking and starting an XATMI program ............................ 307
6.6.1.1 Linking an XATMI program under Windows systems .......... 307
6.6.1.2 Linking an XATMI program under Unix systems ............... 307
6.6.1.3 Linking an XATMI program under BS2000/OSD ................. 308
6.6.1.4 Starting the program ............................................ 308
6.6.2 Setting Environment variables on Windows and Unix systems ... 309
6.6.3 Setting job variables under BS2000/OSD .......................... 310
6.6.4 Trace .................................................................. 312

6.7 Notes on changing from openUTM-Client < V4.0 ..................... 313

6.8 xatmigen messages ....................................................... 314

7 Configuration ............................................................... 317

7.1 Configuration without upicfile ......................................... 318
7.1.1 UPIC-L configuration .................................................... 319
7.1.2 UPIC-R configuration .................................................... 320
7.1.3 Configuration with TNS entries ....................................... 322
7.1.4 Configuration using BCMAP entries .................................. 322

7.2 The side information file (upicfile) ................................... 323
7.2.1 Side information for standalone UTM applications .............. 324
7.2.2 Side information for UTM cluster applications ................... 332
7.2.3 Side information for the local application .......................... 339

7.3 Coordination with the partner configuration ........................... 343
## 8 Implementing CPI-C applications

### 8.1 Runtime environment, linking, starting

- **8.1.1 Implementing in Windows systems**
  - Compilation, linking, starting
  - Runtime environment, environment variables
  - Special features of implementing UPIC local on Windows systems

- **8.1.2 Implementation in Unix systems**
  - Compilation, linking, starting
  - Runtime environment, environment variables
  - Special features of implementing UPIC local on Unix systems

- **8.1.3 Deployment under BS2000/OSD**

### 8.2 Handling of CPI-C partners by openUTM

### 8.3 Behavior in the event of errors

### 8.4 Diagnostics

- **8.4.1 UPIC log file**
- **8.4.2 UPIC trace**
- **8.4.3 PCMX diagnostics (Windows systems)**

## 9 Examples

### 9.1 Sample programs for Windows systems

- **9.1.1 uptac**
- **9.1.2 utp32**
- **9.1.3 tpcall**
- **9.1.4 upmulti**
- **9.1.5 OCTXtestContainer**
- **9.1.6 UpicAutClt**
- **9.1.7 UpicAutSvr**
- **9.1.8 upic-cob**
- **9.1.9 UpicSimpleClient**

### 9.2 Generation UPIC on Windows systems <-> openUTM on BS2000/OSD

- **9.2.1 Generation on the Windows system**
- **9.2.2 Generation on the BS2000 host**

### 9.3 Generation UPIC on Windows systems <-> openUTM on Unix systems

- **9.3.1 Generation on the Windows system**
- **9.3.2 Generation on the Unix system**
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Appendix</td>
<td>381</td>
</tr>
<tr>
<td>10.1</td>
<td>Differences vis à vis the X/Open CPI-C interface</td>
<td>381</td>
</tr>
<tr>
<td>10.2</td>
<td>Character sets</td>
<td>384</td>
</tr>
<tr>
<td>10.3</td>
<td>State table</td>
<td>386</td>
</tr>
<tr>
<td></td>
<td>Glossary</td>
<td>393</td>
</tr>
<tr>
<td></td>
<td>Abbreviations</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>Related publications</td>
<td>431</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>439</td>
</tr>
</tbody>
</table>
1 Preface

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

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

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

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

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

openUTM can be run as a standalone UTM application or on several different computers as a UTM cluster application.
openUTM forms part of the comprehensive openSEAS offering. In conjunction with the Oracle Fusion middleware, openSEAS delivers all the functions required for application innovation and modern application development. Innovative products use the sophisticated technology of openUTM in the context of the openSEAS product offering:

- BeanConnect is an adapter that conforms to Sun's Java Connector Architecture (JCA) and supports standardized connection of UTM applications to J2EE application servers. This makes it possible to integrate tried-and-tested legacy applications in new business processes.

- The WebTransactions member of the openSEAS family is a product that allows tried-and-tested host applications to be used flexibly in new business processes and modern application scenarios. Existing UTM applications can be migrated to the Web without modification.
1.1 Brief description of the openUTM-Client product

The product openUTM-Client offers client/server communication with openUTM server applications which run on Windows systems, Unix systems and BS2000/OSD systems. openUTM-Client is available with the carrier systems UPIC and OpenCPIC. It is the job of the carrier system to establish the connection to other necessary system components (e.g. the transport system) and to control the client/server communication.

For calling the services of an UTM server application, openUTM-Client provides the standardized X/Open interfaces CPI-C and XATMI. CPI-C and XATMI are supported by both the UPIC and the OpenCPIC carrier systems.

- **CPI-C** stands for **Common Programming Interface for Communication**. CPI-C implements a subset of the functions of the CPI-C interface defined in X/Open. CPI-C enables client/server communication between a CPI-C client application and services of a UTM application which use either the CPI-C or the KDCS interface.

- **XATMI** is an X/Open interface for a communication resource manager, with which client/server communication can be implemented with remote UTM server applications. XATMI enables communication with the services of a UTM application which use the XATMI server interface. XATMI is the interface defined in the X/Open Preliminary Specification.

For Windows systems openUTM-Client with the UPIC carrier system also offers the ActiveX control *UpicB.ocx* and the ActiveX automation server *UpicAutSvr.exe*:

- The ActiveX control *UpicB.ocx* simplifies and speeds up the programming of clients – especially when you want to use programming systems such as Visual Basic. *UpicB.ocx* has a COM/DCOM interface.

- The ActiveX automation server *UpicAutSvr.exe* is an external COM server: it enables the advantages of *UpicB.ocx* to also be used from within any other ActiveX-capable application, e.g. from Microsoft Office applications such as Excel.

**openUTM-Client for different platforms**

openUTM-Client is available for the following platforms:
- Windows platforms
- Unix platforms
- BS2000/OSD (UPIC carrier system only)

Because the CPI-C and XATMI interfaces are standardized, i.e. are identical on all platforms, client applications created and tested on one platform can be ported to any of the other platforms.
1.2 Summary of contents and target group

This manual is intended for organization planners, application planners, programmers and administrators who wish to create and run client applications based on UPIC for communication with UTM server applications. It therefore describes openUTM-Client only for the UPIC carrier system. Information on the OpenCPIC carrier system can be found in a separate manual “openUTM-Client for the OpenCPIC Carrier System”.

The description given in this manual applies to all client platforms Windows 2003, Windows XP, Unix platforms, Linux platforms and BS2000/OSD. Special information which relates to a specific platform only is indicated by corresponding headings.

Wherever the term Unix system or Unix platform is used in the following, then this should be understood to mean both a Unix-based operating system such as Solaris or HP-UX and a Linux distribution such as SUSE or Red Hat.

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

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

1.3.1 openUTM documentation

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

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

All the manuals are available as PDF files on the internet at http://manuals.ts.fujitsu.com

On this site, enter the search term “openUTM V6.0“ in the Search by product field to display all openUTM manuals of version 6.0.

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

The following sections provide a task-oriented overview of the openUTM V6.0 documentation. You will find a complete list of documentation for openUTM in the chapter on related publications at the back of the manual on page 431.

Introduction and overview

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

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

- You will require the Creating Applications with X/Open Interfaces manual if you want to use the X/Open interface. This manual contains descriptions of the UTM-specific extensions to the X/Open program interfaces TX, CPI-C and XATMI as well as notes on configuring and operating UTM applications which use X/Open interfaces. In addition, you will require the X/Open-CAE specification for the corresponding X/Open interface.

- If you want to interchange data on the basis of XML, you will need the document entitled openUTM XML for openUTM. This describes the C and COBOL calls required to work with XML documents. This description is available as a PDF document (online).

- For BS2000/OSD there is supplementary documentation on the programming languages Assembler, Fortran, Pascal-XT and PL/1. This is available in the form of PDF files (online).

Configuration

The Generating Applications manual is available to you for defining configurations. This describes how to use the UTM tool KDCDEF to define the configuration and create the KDCFILE for a UTM application. Generation of a UTM cluster application is also described. 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.

Linking, starting and using UTM applications

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

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

- If you are using **openUTM WinAdmin**, the graphical administration workstation, the following documentation is available to you:
  - A **description of WinAdmin**, which provides a comprehensive overview of the functional scope and handling of WinAdmin. This document is shipped with the software and is also available online as a PDF file.
  - The **online help system**, which provides context-sensitive help information on all dialog boxes and associated parameters offered by the graphical user interface. In addition, it also tells you how to configure WinAdmin in order to administer and generate openUTM applications.

Testing and diagnosing errors

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

Creating openUTM clients

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

- The **openUTM-Client for the UPIC Carrier System** describes the creation and operation of client applications based on UPIC. In addition to the description of the CPI-C and XATMI interfaces, you will find information on how you can use the C++ classes or ActiveX to create programs quickly and easily.

- The **openUTM-Client for the OpenCPIC Carrier System** manual describes how to install and configure OpenCPIC and configure an OpenCPIC application. It describes how to install OpenCPIC and how to configure an OpenCPIC application. It indicates what needs to be taken into account when programming a CPI-C application and what restrictions apply compared with the X/Open CPI-C interface.
The documentation for the **JUpic-Java classes** shipped with BeanConnect is supplied with the software. This documentation consists of Word and PDF files that describe its introduction and installation and of Java documentation with a description of the Java classes.

The **BizXML2Cobol** manual describes how you can extend existing COBOL programs of a UTM application in such a way that they can be used as an XML-based standard Web service. How to work with the graphical user interface is described in the **online Help system**.

If you want to provide UTM services on the Web quickly and easily then you need the manual **WebServices for openUTM**. The manual describes how to use the software product WS4UTM (WebServices for openUTM) to make the services of UTM applications available as Web services. The use of the graphical user interface is described in the corresponding **online help system**.

**Communicating with the IBM world**

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

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

Integrating J2EE application servers and UTM applications

The BeanConnect adapter forms part of the openSEAS product suite. The BeanConnect adapter implements the connection between conventional transaction monitors and J2EE application servers and thus permits the efficient integration of legacy applications in Java applications.

- The manual BeanConnect describes the product BeanConnect, that provides a JCA 1.5-compliant adapter which connects UTM applications with applications based on J2EE, e.g. the Oracle application server.
  The manuals for the Oracle application server can be obtained from Oracle.

Connecting to the web and application integration

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

The manuals will also be supplemented by JavaDocs.
1.3.3 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.

- **BS2000/OSD:**
  
  On a BS2000 computer, you will find information in the Release Note (file name SYSFGM.UTM-Client.060.E and possibly in a README file as well (file name SYSRME.UTM-Client.060.E). Please ask your systems support for the user ID on which the README file is located. You can view the README file with the /SHOW-FILE command or in an editor or you can print it to a standard printer with the following command:

  ```
  /PRINT-DOCUMENT filename.LINE-SPACING=*BY-EBCDIC-CONTROL
  ```

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

- **Windows systems:**
  
  The README file and any other files, such as a manual supplement file, can be found in the `utmpath` under `\Docs\language`. 
1.4 Changes since the last version of this manual

The manual openUTM-Client V6.0 for the UPIC Carrier System covers the following new features introduced since the manual openUTM-Client V5.3 for the UPIC Carrier System:

Communication with UTM cluster applications

As of openUTM-Client V6.0, it is possible to apply static load distribution for jobs sent from UPIC clients to a UTM cluster application. If this is done, the jobs are distributed across the individual node applications of the UTM cluster application. To do this, it is necessary to configure an openUTM cluster. One node application with which the next UPIC communication operation is performed is arbitrarily selected from a list of node applications.

The following interfaces were changed to support this:

● **upicfile**: New prefix CD:

You configure an openUTM cluster by creating one entry for each node application of a UTM cluster application with this prefix. You can only make use of static distribution if you choose a symbolic destination name with the prefix CD when calling `Initialize_Conversation`. If the configuration is not correct, the signon attempt is rejected with the return code CM_PROGRAM_PARAMETER_CHECK.

If you wish to administer the UTM cluster application using a UPIC client, it is recommended that you do not address the application using a symbolic destination name of an openUTM cluster, as you do not have the opportunity to explicitly select one of the node applications if you do this.

● **CPI-C interface**:

  – New return codes for `Receive` and `Receive_Mapped_Data`:

    CM_SECURITY_USER_GLOBALLY_UNKNOWN
    CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
    CM_SECURITY_TRANSIENT_ERROR

  – The partner address cannot be set in the program. The return code CM_CALL_NOT_SUPPORTED is therefore returned if `Set_Partner_Host_Name`, `Set_Partner_IP_Address` or `Set_Partner_LU_Name` is called.

Other changes

If UPIC attempts to sign on to a UTM application in BS2000/OSD using a user ID generated with a Kerberos principal, this is rejected with the return code CM_SECURITY_NO_KERBEROS_SUPPORT.
1.5 Notational conventions

Symbols

Parts of the description which only apply to specific UPIC platforms are indicated by a symbol in the left margin as follows:

- \text{B} Indicates parts of the description that are only significant for UPIC under BS2000/OSD.
- \text{X} Indicates parts of the description that are only significant for UPIC under Unix systems.
- \text{W} Indicates parts of the description that are only significant for UPIC under Windows systems.
- \text{B/X} Indicates parts of the description that are only significant for UPIC under BS2000/OSD and Unix systems.
- \text{B/W} Indicates parts of the description that are only significant for UPIC under BS2000/OSD and Windows systems.
- \text{X/W} Indicates parts of the description that are only significant for UPIC under Unix and Windows systems.

Other symbols

- \text{\textdegree} Indicates references to comprehensive, detailed information on the relevant topic.
- \text{i} Indicates notes that are of particular importance.
- \text{!} Indicates warnings.
## Metasyntax

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

<table>
<thead>
<tr>
<th>Representation</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPPERCASE LETTERS</strong></td>
<td>Uppercase letters denote constants (names of calls, statements, field names, commands and operands etc.) that are to be entered in this format.</td>
<td>LOAD-MODE=STARTUP</td>
</tr>
<tr>
<td><strong>lowercase letters</strong></td>
<td>In syntax diagrams and operand descriptions, lowercase letters are used to denote place-holders for the operand values.</td>
<td>KDCFILE=filebase</td>
</tr>
<tr>
<td><strong>lowercase letters in italics</strong></td>
<td>In running text, variables, the names of data structures and fields, and keywords (e.g. C commands, Unix and Windows file names etc.) are indicated by lowercase letters in italics.</td>
<td>COBOL data structures are defined in the COPY member KCINIC; C/C++ data structures are defined in the header file kcini.h.</td>
</tr>
<tr>
<td>**{ } and</td>
<td>Curly brackets contain alternative entries, of which you must choose one. The individual alternatives are separated within the curly brackets by pipe characters.</td>
<td>STATUS={ ON</td>
</tr>
<tr>
<td>[ ]</td>
<td>Square brackets contain optional entries that can also be omitted.</td>
<td>KDCFILE=( filebase [, { SINGLE</td>
</tr>
<tr>
<td>()</td>
<td>Where a list of parameters can be specified for an operand, the individual parameters are to be listed in parentheses and separated by commas. If only one parameter is actually specified, you can omit the parentheses.</td>
<td>KEYS=(key1,key2,...,keyn)</td>
</tr>
<tr>
<td><strong>Underscoring</strong></td>
<td>Underscoring denotes the default value.</td>
<td>CONNECT= { A/YES</td>
</tr>
<tr>
<td><strong>abbreviated form</strong></td>
<td>The standard abbreviated form of statements, operands and operand values is emphasized in boldface type. The abbreviated form can be entered in place of the full designation.</td>
<td>TRANSPORT-SECTOR=c‘C’</td>
</tr>
</tbody>
</table>

Table 1: Metasyntax
<table>
<thead>
<tr>
<th>Representation</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
</table>
| ...            | An ellipsis indicates that a syntactical unit can be repeated. It can also be used to indicate sections of a program or syntax description etc. | Start KDCDEF  
:  
:  
OPTION DATA=statement_file  
:  
END |

Table 1: Metasyntax
2 Application area

Since the screen layout is not actually a function of the transaction monitor, it is delegated to clients by the UTM application. The UTM application is thus the server. openUTM-Client with the interfaces CPI-C and XATMI allows you to create client programs that work with the UTM application as the server.

However, you can also use client programs as the request driver.

The client/server concept

The aim of the client/server concept is to provide the individual users in a network with services (such as data, programs, devices) and to ensure that optimum use is made of the strong points of the individual systems.

The client/server concept is always implemented where many clients require the same service. An analogy to the client/server concept is as follows: the procedure or subroutine call sets up a client/server relationship between the main program and the subroutine. The only difference is that the called procedure now runs remotely from the “client”.

Clients (users of services) can request services and information from all servers in the network.

Servers (providers of services) provide services whereby shared information sources, such as files and databases, can be distributed randomly within a network configuration.
2.1 The concept of openUTM-Client

To call services, openUTM-Client offers standardized X/Open interfaces on various platforms and carrier systems.

![Diagram of openUTM-Client interfaces and carrier systems]

<table>
<thead>
<tr>
<th>openUTM-Client Interfaces</th>
<th>Carrier system</th>
<th>Operating system</th>
</tr>
</thead>
</table>
| CPI-C                     | UPIC or OpenCPIC | Unix system MS-Windows BS2000/OSD

1 BS2000/OSD only with UPIC carrier system

Figure 1: Standardized X/Open interfaces

**Interfaces**

openUTM-Client can be programmed with the X/Open interfaces CPI-C and XATMI.

- **Concealed CPI-C interface for Windows and Unix systems with UPIC carrier system:**
- For Unix systems and Windows systems, openUTM-Client (UPIC carrier system) provides a concealed version of the CPI-C interface: the CUpic wrapper class.
- A worker thread is created for each CUpic object. In this way, several UPIC conversations can be active in parallel in an application. The threads are created and controlled transparently by the CUpic class.
- For more information on the CUpic class, see chapter “C++ class CUpic” on page 69.
Additionally for Windows systems

For Windows systems, openUTM-Client (UPIC carrier system) provides a concealed version of the CPI-C interface: the ActiveX control element UpicB.ocx.

UpicB.ocx facilitates the programming of clients - especially if you use programming environments such as Visual Basic.

The automation server UpicAutSvr.exe enables the advantages offered by UpicB.ocx to be used from within any ActiveX-capable application, e.g. from Microsoft Office applications such as Excel.

For more information on the CUpic class, see chapter “ActiveX control UpicB.ocx” on page 35.

Carrier systems

The CPI-C and XATMI interfaces are provided by both the UPIC carrier system and the OpenCPIC carrier system. The task of the carrier system is to establish the connection to the other necessary components, such as the transport access system (TCP/IP in Windows, Unix systems or BS2000/OSD, PCMX-32 in Windows systems, CMX/PCMX in Unix systems or BCAM in BS2000/OSD).

The UPIC carrier system offers the following advantages over OpenCPIC:

● The client program can simulate the activation of function keys.
● Format IDs can also be exchanged between client and server as structure information together with the data.
● The client program can assign a new password.

Operating system platforms

A carrier system can reside on the following different kinds of different platform:

● Windows systems
● all common Unix systems
● BS2000/OSD systems (UPIC carrier system only)

Because the CPI-C and XATMI interfaces are standardized, i.e. identical on all platforms, the client applications created and tested on one platform can be ported to any of the other platforms.
Definition of terms

A program containing CPI-C calls is referred to below as a **CPI-C program** and a program containing XATMI calls is referred to as an **XATMI program**. The underlying carrier system is only mentioned if it influences the functionality or is visible on the interface.

A **CPI-C application** or an **XATMI application** is the totality of the CPI-C or XATMI programs plus all configuration files required for the respective carrier system.
2.2 Client/server communication with openUTM

The diagram below indicates the interfaces via which openUTM clients can communicate with a UTM server.

![Diagram showing interfaces between openUTM server and openUTM Clients]

Figure 2: Interfaces between openUTM server and openUTM Clients

A client with a CPI-C program can communicate both with a KDCS program unit and with a CPI-C program unit; a client with an XATMI program can only ever use an XATMI program unit as a service. A KDCS program unit is a program unit of a UTM server which contains KDCS calls.

On all platforms, the client and server can reside on the same system.

A UTM server application is always referred to below as a UTM application, or simply as UTM.
2.3 UPIC local, UPIC remote and multithreading

With UPIC as the carrier system, you have two main options for linking client programs: UPIC local (Unix systems/Windows systems) and UPIC remote (all platforms).

Unless otherwise specified, the information in this manual applies to both variants.

X/W | UPIC local (Unix systems/Windows systems)
X/W With UPIC local (UPIC-L), you can link a client program locally with a UTM application on the same Unix system or Windows system. The UPIC-local carrier system is available for Unix systems and Windows systems. It is integrated into the openUTM server software. For connection via UPIC local you therefore require neither the product openUTM-Client nor the communication components CMX or PCMX.
X/W This option is only available on a Unix system or a Windows system.

X/W

<table>
<thead>
<tr>
<th>Unix system or Windows system</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPIC program</td>
</tr>
<tr>
<td>Interface functions</td>
</tr>
<tr>
<td>CPI-C / XATMI calls</td>
</tr>
<tr>
<td>UTM application</td>
</tr>
</tbody>
</table>

Figure 3: Local connection to a UTM application

X/W The interface functions provide a user-friendly interface. The client program communicates with the UTM application using CPI-C calls or XATMI calls, whereby only net data is transmitted.
UPIC remote

With UPIC remote (UPIC-R) you can link a client program with UTM applications running on any system in the network. This option is available for all server platforms (Windows systems, Unix systems and BS2000/OSD). You need the product openUTM-Client for this. openUTM-Client contains two different versions of UPIC remote. The current variant (from V5.1 onwards) uses TCP/IP via the socket interface. No additional communications components are necessary for this. In the classic variant, access to the network is controlled via the platform-specific communication components PCMX or CMX (see figure 4).

Figure 4: Remote connection to UTM applications
With a remote connection too, it is possible that the client program and the UTM application reside on the same system. Even in this case, however, communication between the client program and openUTM is handled by the communication components TCP/IP, PCMX or CMX.

**Multithreading**

The UPIC carrier system is basically multithreading-capable. Two components determine whether you can use this capability in your application:

- The operating system must support multithreading
- The communication system used must support multithreading

The situation with UPIC on the specific platforms is as follows:

- **X/W** UPIC-L is not multithreading-capable
- **W** UPIC-R on Windows systems is multithreading-capable without restrictions
- **X** UPIC-R on Unix systems is multithreading-capable
- **B** UPIC-R on BS2000/OSD systems is not multithreading-capable

See the corresponding release notice for the precise details.
2.4 Support for UTM cluster applications

An openUTM client with UPIC as the carrier system can communicate with a UTM cluster application in the same way as with a standalone UTM application.

A cluster is a number of computers (nodes) connected over a fast network. openUTM runs on a cluster in the form of a UTM cluster application. From a physical perspective, a UTM cluster application is made up of several identically generated UTM applications (the node applications) that run on the individual nodes.

The client requires a list of the associated node applications. An arbitrary node application is then selected from this list to be used for the next communication operation.

If communication is not possible with the selected node application, the system automatically attempts to establish a connection to the next node application in the list. This process is repeated until communication can be successfully established to a running node application or until the system detects that none of the node applications in the list can be accessed.

Bild 5: Communication with a UTM cluster application

The list of node applications for each UTM cluster application is passed in the side information file (upicfile). For details, see the section “Side information for UTM cluster applications” on page 332.
3 ActiveX control UpicB.ocx

The ActiveX control UpicB.ocx enables you to create openUTM-Client applications quickly and easily. With UpicB.ocx you do not need to integrate any CPIC calls into the source code of your client program. Instead, you use the properties and methods provided for you by UpicB.ocx.

Calls such as Enable_UTM_UPIC, Initialize_Conversation or Allocate are superfluous with UpicB.ocx: the CPI-C interface of the UPIC carrier system is concealed by the COM/DCOM interface of UpicB.ocx.

The COM/DCOM interface of UpicB.ocx is equivalent to the general object model COM (Component Object Model) which has become established as the Microsoft standard. This model defines a procedure for the way in which software components make their services available to other software components. DCOM (Distributed COM) enables this standardized form of binary interaction of components between different hosts as well.

Thanks to its COM/DCOM interface, UpicB.ocx can be used from within any program which allows connection of COM/DCOM or OLE components. It is, for example, ideally suited for use with Microsoft programming systems like Visual Basic (see example on page 38) or Visual C++ or Borland Delphi.

With the ActiveX automation module UpicAutSrv.exe - an external COM server which, like UpicB.ocx, is supplied together with openUTM-Client - the services of UpicB.ocx can however also be used within standard Windows applications such as Microsoft Office. So for Excel diagrams or standard Word letters, for example, you can implement automatic access to data which is provided by UTM server applications.
Although with *UpicB.ocx* you are not directly using the X/Open standard program interface, the compatibility is retained through the COM concept. The interfaces released by the ActiveX control *UpicB.ocx* cannot be modified, according to the COM conventions. They are identified via IDs which are globally unique.

Changes that come from extensions or further developments do not affect these interfaces: in such cases, rather than the existing interfaces being modified, new COM interfaces are released with their own IDs. Your clients that are programmed with *UpicB.ocx* will therefore also be supported by future versions of openUTM-Client without conflict.
The main advantages of *UpicB.ocx* are as follows:

- simple and convenient handling through embedding in the Microsoft architecture, e.g. entry of default values via property pages)
- option of starting non-blocking receive services and automatic activation of program segments when data is received (event-driven processing)
- establishment and use of several parallel conversations
- it can be used in MS Transaction Server
- it can be used in MS Active Server pages
- it can be used in Java programs containing a connection to the COM/DCOM architecture (currently only possible with MS-Java)
Example: using UpicB.ocx in Visual Basic

During installation of openUTM-Client, UpicB.ocx is only copied onto the hard disk. You must create an entry in the Windows registry so UpicB.ocx will be known throughout the system. The Microsoft Regsvr32.exe utility is used for this purpose and it is copied into the installation directory during installation. The Register UPIC menu item is available in the openUTM-Client program group in order to facilitate this task. Clicking this menu item causes UpicB.ocx to be registered.

If you open the “Upic OCX” register in Visual Basic V6.0 (e.g. via the Components... command in the Project menu or by pressing CTRL+T), UpicB.ocx will be offered to you in the selection list as “UpicB OLE Control module”:

When you have selected “UpicB OLE Control module” and clicked on OK, the following icon will appear in the Toolbox:
Like every other control element offered in the Toolbox you can now integrate *UpicB.ocx* into your Visual Basic form by clicking on the icon and then setting up a corresponding area in your form.

The UPIC area displayed is only visible at design time, not at runtime. It indicates that the properties and methods of *UpicB.ocx* are now available in the form.

You can now, for example, add buttons to your form and call up *UpicB.ocx* methods in the corresponding code window. The section “Application example” on page 64 contains examples of such basic code.

Via *UpicB.ocx*’s Property Pages window you can select defaults for the *UpicB.ocx* properties, e.g. for the name of the desired UTM service. You then do not need to code these properties explicitly.

The Property Pages can be opened in various ways, e.g. by clicking with the right mouse button in the UPIC area and choosing the “Properties...” command. For diagrams of the individual property pages, see section “Property Pages window” on page 62.

A complete example of a Visual Basic client application which uses *UpicB.ocx* is delivered with openUTM-Client. It is stored under the following path:

```
<installation_directory>\samples\UpicCOM\OCXTestContainer
```

(see also section “OCXTestContainer” on page 375).
3.1 Interfaces of UpicB.ocx

To use UpicB.ocx you do not need any detailed knowledge of the CPI-C interface as a rule. You only need to familiarize yourself with the properties and methods that UpicB.ocx provides (See section “Properties of UpicB.ocx” on page 41 and section “Methods of UpicB.ocx” on page 48).

If you are working with, say, Visual Basic, you need only open up the UPIC icon from the toolbox with the mouse in a Visual Basic form. The UpicB.ocx control element together with its interfaces and their description are thus made known to the program and can now be used.

Details for interested readers:

The following information (up to the beginning of section “Properties of UpicB.ocx” on page 41) is intended for interested users who want to find out more.

UpicB.ocx makes its services available via the following COM/DCOM interfaces:

- **_DUpicB:**
  
  _DUpicB is a dispatch interface. In accordance with the COM concept it has a globally unique identifier (GUID): {9F172F36-A117-11D0-A347-0000E8A39D41}. All properties and methods of the control element are accessible via this interface.

- **_DUpicBEEvents:**
  
  _DUpicBEEvent is also a dispatch interface. It has the ID {9F172F37-A117-11D0-A347-0000E8A39D41}. As well as the properties and methods, this interface also provides the DataReceived event, which makes non-blocking receive services possible.

UpicB.ocx is addressed by client programs via the program ID “UPIC.UpicCtrl.2”. The corresponding class ID is {9F172F38-A117-11D0-A347-0000E8A39D41}, which like the IDs of the interfaces is a GUID.

The resetting of the properties and the calling of the methods is protected internally by a semaphore against simultaneous multiple setting or calling. At any one time, either one property may be reset or one method may be called within a thread. If the apartment model is used as a storage model, a UPIC object can only be used within one thread. Properties and methods cannot be called from within another thread, although more than one UPIC object can exist within a thread.
3.1.1 Properties of UpicB.ocx

The properties of UpicB.ocx can be roughly divided into two groups:

- One group describes the basic conversation characteristics, i.e. settings for the communication relationship between an openUTM-Client program and a UTM service. These are the properties LocalName, TPName, UserID, Password, SymDestName.

- The second group consists of properties which describe the individual send or receive calls in more detail. These are the properties FunctionKey, MapOn, MapName, BlockingReceive, ReceiveTimer.

This leaves us with the property Trace for activating/deactivating the UpicB.ocx trace, which belongs to neither of the above categories.

The following table provides an overview of the properties of UpicB.ocx. The individual properties are described in more detail on page 43.

<table>
<thead>
<tr>
<th>Property</th>
<th>Data type</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>FunctionKey</td>
<td>long</td>
<td>Set function key</td>
</tr>
<tr>
<td>LocalName</td>
<td>BSTR</td>
<td>Specify local name of UPIC program</td>
</tr>
<tr>
<td>TPName</td>
<td>BSTR</td>
<td>Set transaction code</td>
</tr>
<tr>
<td>UserID</td>
<td>BSTR</td>
<td>Set UTM user ID</td>
</tr>
<tr>
<td>Password</td>
<td>BSTR</td>
<td>Set password</td>
</tr>
<tr>
<td>MapOn</td>
<td>Boolean</td>
<td>Show exchange of format names</td>
</tr>
<tr>
<td>MapName</td>
<td>BSTR</td>
<td>Specify format names</td>
</tr>
<tr>
<td>BlockingReceive</td>
<td>Boolean</td>
<td>Select blocking or non-blocking receive</td>
</tr>
<tr>
<td>ReceiveTimer</td>
<td>long</td>
<td>Set timeout timer</td>
</tr>
<tr>
<td>SymDestName</td>
<td>BSTR</td>
<td>Specify reference to addressing information</td>
</tr>
<tr>
<td>Trace</td>
<td>BSTR</td>
<td>Activate or deactivate UPIC control trace</td>
</tr>
<tr>
<td>PartnerLUName</td>
<td>BSTR</td>
<td>Set partner_LU_name</td>
</tr>
<tr>
<td>PartnerPort</td>
<td>long</td>
<td>Specify port number of partner application</td>
</tr>
</tbody>
</table>

You can use UpicB.ocx in all programming languages which support COM. You must remember that the methods for setting properties may be different. For example, in Visual Basic you modify a property by making a simple assignment in the form object.property=value, in Visual C++ you modify a property using the properties dialog box or the relevant SetProperty() function.
Rules for setting properties

There are rules as to when a property can be set. These rules are given in the descriptions of the individual properties. The rules prevent, for example, the name of the UTM service (TPName) from being changed between the sending of the data and the receiving of the result.

Basically, the following applies:

- Properties which define conversation characteristics can only be reset at the following times:
  - before the first SendMsg or Restart call or
  - between a Disconnect call and the following SendMsg or Restart call.

- Properties which describe individual calls in more detail and the Trace property can be reset at any time, except when a non-blocking receive service is in progress.

Behavior in the event of incorrect input

Except with FunctionKey, ReceiveTimer and Trace, incorrect entries for properties are not corrected by UpicB.ocx, but are passed on to the corresponding UPIC functions. If the values for FunctionKey and ReceiveTimer are invalid, these properties are always set to the initialization value. If the value for the Trace property is invalid, it is set to an empty string.
**FunctionKey - set function key**

The `FunctionKey` property specifies a function key of the UTM partner application. The value of `FunctionKey` is sent to the UTM application the next time data is sent, and the return code that is assigned to the corresponding function key in the UTM application is then executed. The client program has then in effect “pressed the function key”.

You can select the desired function key with the mouse in a Property Pages window, and the corresponding value (type `long`) is then set automatically.

This property can be reset at any time except when a non-blocking receive service is in progress. For `FunctionKey` to be effective, it should be set before a `SendMsg`.

If an invalid value is entered, `UpicB.ocx` corrects the value to the initialization value 0 (CM_UNMARKED).

**Data type:** `long`

**Value range:**
- CM_UNMARKED
- CM_FKEY_K1 ... CM_FKEY_K14
- CM_FKEY_F1...CM_FKEY_F24

**Initialization:** 0 (CM_UNMARKED)

**LocalName - specify local name of UPIC program**

Local name under which the client program signs on to the UPIC carrier system.

The following specifications are possible:
- name under which the program is entered in the TNS directory
- local name as entered in the `upicfile`
- empty string. (In this case the DEFAULT name defined in the `upicfile` is used.)

This property can only be reset before the first `SendMsg` or `Restart` call or between a `Disconnect` call and the next `SendMsg` or `Restart` call.

**Data type:** `BSTR`

**Value range:** `LocalName` must not be longer than 8 characters and must not contain any blanks. An empty `LocalName` means the DEFAULT name is used.

**Initialization:** Empty string
**TPName - set transaction code**

In *TPName* you enter the transaction code of the desired UTM service.

This property can only be reset before the first *SendMsg* or *Restart* call or between a *Disconnect* call and the next *SendMsg* or *Restart* call.

- **Data type:** BSTR
- **Value range:** *TPName* must be between 1 and 8 characters long and must not contain any blanks.
- **Initialization:** Empty string

**UserID - set UTM user ID**

With *UserID* you specify a user ID for a UTM application. If *UserID* is blank, the security type of the conversation is set to CM_SECURITY_NONE, otherwise it is automatically given the value CM_SECURITY_PROGRAM.

This property can only be reset before the first *SendMsg* or *Restart* call or between a *Disconnect* call and the next *SendMsg* or *Restart* call.

- **Data type:** BSTR
- **Value range:** A user ID can be between 0 and 8 characters long. If the length is 0, no user ID is used to establish the conversation.
- **Initialization:** Empty string

**Password - set password**

In *Password* you specify the password of a UTM user ID. This property is only interpreted if a user ID is set in *UserID*.

This property can only be reset before the first *SendMsg* or *Restart* call or between a *Disconnect* call and the next *SendMsg* or *Restart* call.

- **Data type:** BSTR
- **Value range:** A password can be between 0 and 8 characters long.
- **Initialization:** Empty string
MapOn - show exchange of format names

*MapOn* indicates whether the *MapName* property is currently set, i.e. whether a format name is to be sent with the data.

This property can be reset at any time except when a non-blocking receive service is in progress.

- **Data type:** Boolean
- **Initialization:** FALSE (i.e. no format names are to be transferred)

MapName - specify format name

*MapName* contains the format name that is to be sent or was received.

This property can be reset at any time except when a non-blocking receive service is in progress.

- **Data type:** BSTR
- **Initialization:** Empty string

BlockingReceive - select blocking or non-blocking receive

With *BlockingReceive* you define whether within a conversation the *ReceiveMsg* method is executed with blocking or without.

If the value TRUE is set, the *Receive* method is called with blocking, i.e. the program waits in the “Receive” state until information arrives.

If the value FALSE is set, the *Receive* method is called without blocking, i.e. the program does not wait but is continued immediately - even if there is no information yet.

Regardless of the value in *BlockingReceive*, *UpicB.ocx* moreover offers the option of starting a non-blocking receive service with the *RequestForData* method (see page 52).

- **Data type:** Boolean
- **Initialization:** TRUE
Properties of UpicB.ocx

**ReceiveTimer - set timeout timer**

With **ReceiveTimer** you set the timeout timer (number of milliseconds). If this is set and **BlockingReceive** equals TRUE, **ReceiveMsg** calls are aborted when the specified time interval expires. In the case of non-blocking receive services, the **DataReceived** event is triggered when the timer expires.

The value 0 means that no timer is set: in this case, therefore, blocking waiting occurs.

This property can be reset at any time except when a non-blocking receive service is in progress.

If an incorrect value is entered, **UpicB.ocx** corrects it to the initialization value.

- **Data type:** long
- **Value range:** integer greater than or equal to 0 (number of milliseconds)
- **Initialization:** 0

**SymDestName - specify reference to addressing information**

In **SymDestName** you specify the name under which the addressing of the partner is entered in the **upicfile**. If an empty string is set, the DEFAULT name is used (can be entered in the **upicfile** with .DEFAULT).

An empty string must be set in operation without a **upicfile**.

This property can only be reset before the first **SendMessage** or **Restart** call or between a **Disconnect** call and the next **SendMessage** or **Restart** call.

- **Data type:** BSTR
- **Value range:** **SymDestName** is 8 characters long
- **Initialization:** Empty string
**Trace - activate or deactivate UPIC control trace**

This property is used to activate the UPIC control trace.

The trace is activated if this property contains the name of a valid directory to which the trace file for this object is recorded. The name of a trace file has the following format:

UpicT<timestamp in yyyymmddhhmmss>_##.uct

where ## is a number up to two digits.

The writing of trace files can be deactivated by reassigning an empty string to the property.

Plain text is written to the UPIC control trace file, so the file must be read in a text editor such as Notepad.

If an incorrect value is entered, UpicB.ocx corrects it to an empty string.

This property can be reset at any time except when a non-blocking receive service is in progress.

**Data type:** BSTR

**Initialization:** The property is preset to the value entered in the Registry:

HKEY_CLASSES_ROOT\CLSID\{9F172F38-A117-11D0-A347-0000E8A39D41}\InprocServer32\Trace

If this entry is not present, the value is set to an empty string.

**PartnerLUName – specify addressing information**

You address the partner in PartnerLUName. You must specify the PartnerLUName in two levels separated by a dot ("."). The part to the left of the dot is the application name, the part to the right is the system. The dot itself is not part of the address.

This property can be reset only before the first call of SendMsg or Restart or between a Disconnect call and a subsequent SendMsg or Restart call.

**Data type:** BSTR

**Value range:** PartnerLUName is a string up to 32 bytes long.

**Initialization:** Empty string
**PartnerPort**

You set the port number of the partner application using `PartnerPort`.

This property can be reset only before the first call of `SendMsg` or `Restart` or between a `Disconnect` call and a subsequent `SendMsg` or `Restart` call.

- **Data type:** `long`
- **Value range:** Integer between 0 and 32676
- **Initialization:** 102

### 3.1.2 Methods of UpicB.ocx

The following table gives an overview of the methods provided by `UpicB.ocx`. These methods are described in more detail in the pages that follow.

<table>
<thead>
<tr>
<th>Method</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SendMsg</code></td>
<td>Send data to a UTM service</td>
</tr>
<tr>
<td><code>ReceiveMsg</code></td>
<td>Receive data from a UTM service</td>
</tr>
<tr>
<td><code>RequestForData</code></td>
<td>Start a non-blocking receive service</td>
</tr>
<tr>
<td><code>PrepareToReceive</code></td>
<td>Change from “Send” to “Receive” state</td>
</tr>
<tr>
<td><code>Restart</code></td>
<td>Initiate restart</td>
</tr>
<tr>
<td><code>Disconnect</code></td>
<td>Sign off from the UPIC carrier system</td>
</tr>
<tr>
<td><code>InConversation</code></td>
<td>Check that the conversation is still running</td>
</tr>
<tr>
<td><code>InSendingState</code></td>
<td>Query right to <code>SendMsg</code> call</td>
</tr>
<tr>
<td><code>IncompleteDataReceived</code></td>
<td>Check completeness of received data</td>
</tr>
<tr>
<td><code>GetLastError</code></td>
<td>Query UPIC error code and error information</td>
</tr>
<tr>
<td><code>ConvertIncoming</code></td>
<td>Convert EBCDIC to local code</td>
</tr>
<tr>
<td><code>ConvertOutgoing</code></td>
<td>Convert local code to EBCDIC</td>
</tr>
</tbody>
</table>
SendMsg - send data to a UTM service

With this method a program sends data to a UTM service. SendMsg can be executed when the InSendingState method returns the value TRUE, e.g. at the start of a conversation in order to establish it, or if, say, the right to call the SendMsg function was transferred in a ReceiveMsg call. If no conversation exists yet, one is automatically established. The characteristics of this conversation are determined by the different properties.

If the MapOn property is set to TRUE, the format name set in MapName is also sent.

Parameters

1st parameter: Unstructured data or basic strings which are sent.

C++ notation: const VARIANT FAR& Data

where:
- Data.vt equals (VT_ARRAY | VT_UI1) and the dimension of Data equals 1 or
- Data.vt equals VT_BSTR.

2nd parameter: Index (beginning with 0) in the data from which point sending begins.

C++ notation: long StartAt

3rd parameter: Length of the data that is sent.

C++ notation: long SendLength

Return value

Type: BOOL

This method returns the value TRUE if the data was sent successfully.
If FALSE is returned, further information can be obtained via the GetLastError method.

Example in Visual Basic

Dim Data As String
Dim rc As Boolean
...
rc = Upic1.SendMsg(Data, 0, Len(Data))
**ReceiveMsg - receive data from a UTM service**

With this method a program receives data from a UTM service.

Before this method is called, one of the following methods must have been called successfully: `SendMsg`, `RequestForData`, `ReceiveMsg`, `PrepareToReceive` or - if necessary - the `Restart` method.

When `ReceiveMsg` is called, the program loses the right to call the `SendMsg` function. However, it can regain this right after returning from `ReceiveMsg`. The `InSendingState` method provides information on this.

If the `MapOn` property is set to TRUE, the `MapName` property contains the format name which has just been received.

**Parameters**

1st parameter: Data buffer for unstructured data or basic strings which are received. The data buffer must be provided by the caller.

C++ notation: `const VARIANT FAR* Data`

where:
- `Data.vt` equals (VT_ARRAY | VT_UI1) and the dimension of Data equals 1 or
- `Data.vt` equals VT_BSTR.

2nd parameter: Index (beginning with 0) in the data buffer as of which the received data is written. Beginning of the data area in the data buffer which serves to receive data.

C++ notation: `long StartAt`

3rd parameter: Length of the data area in the data buffer which serves to receive the data.

C++ notation: `long MaxReceiveDataLength`

4th parameter: Length of the data actually received.

C++ notation: `long *ReceivedDataLength`
Return value

Type: BOOL

This method returns the value TRUE if data was received successfully. If FALSE is returned, further information can be obtained via the GetLastError method.

- If GetLastError returns the value 28 (=CM_UNSUCCESSFUL) and the BlockingReceive property was set to FALSE (i.e. a non-blocking call), no data was yet present when ReceiveMsg was called. ReceiveMsg may have to be called again.

- If GetLastError returns the value 35 (=CM_OPERATION_INCOMPLETE), the BlockingReceive property was set to TRUE and the ReceiveTimer property has a value greater than 0, operation is aborted by the timeout timer.

If the conversation for the ReceiveMsg call was shut down, the InConversation method delivers the value FALSE.

Example in Visual Basic

Dim DataBuf As String *100
Dim Data As Variant
Dim length As Long
Dim rc As Boolean
...
Data = DataBuf 'initialize Variant Buffer...
...
rc = Upic1.ReceiveMsg(Data, 0, Len(Data), length)
**RequestForData - start non-blocking receive service**

With this method a program starts a non-blocking receive service. Before this method is called, one of the following methods must have been called successfully: `SendMsg`, `ReceiveMsg`, `PrepareToReceive` or - if necessary - the `Restart` method.

The `RequestForData` method returns immediately once the service has started. As soon as information for the client program arrives, the `DataReceived` event is triggered and the client program can fetch the information with a `ReceiveMsg` call. Normally this is the receiving data. But the `DataReceived` event is also triggered if an error has occurred or the timer has expired: the `ReceiveMsg` call then delivers the value FALSE and the client program can find out about the error by calling `GetLastError`.

Between the call of the `RequestForData` method and the call of the `ReceiveMsg` method it is not possible to change a `UpicB.ocx` property or successfully call a `UpicB.ocx` method.

**Parameters**

1st parameter: Maximum length of a data area which is required to receive the data. The length must correspond to the value specified at the following `ReceiveMsg` call in `MaxReceiveDataLength`.

C++ notation: long MaxReceiveDataLength

**Return value**

Type: BOOL

This method returns the value TRUE if the non-blocking receive service was started successfully. If FALSE is returned, further information can be obtained via the `GetLastError` method.

**Example in Visual Basic**

```vba
Dim DataBuf As String *100
Dim Data As Variant
Dim length As Long
Dim rc As Boolean
...
Data = DataBuf 'initialize Variant Buffer...
...
rc = Upic1.RequestForData(Len(Data))
...
```

If the "DataReceived" event is received:

```vba
rc = Upic1.ReceiveMsg(Data, 0, Len(Data), length)
```
**PrepareToReceive - change from “Send” to “Receive” state**

With `PrepareToReceive`, the program relinquishes the right to call the `SendMsg` method, and changes to the “Receive” state. The next method to be called must therefore be either `RequestForData` or `ReceiveMsg`.

Before the conversation switches to the “Receive” state, any data still present locally is sent to the UTM application. The method thus contains a “flush” for send data.

**Parameters**

This method has no parameters.

**Return value**

Type: BOOL

This method returns the value TRUE if the program has switched to the “Receive” state. If FALSE is returned, further information can be obtained via the `GetLastError` method.

**Example in Visual Basic**

```vbnet
Dim DataBuf As String *100
Dim Data As Variant
Dim length As Long
Dim rc As Boolean
...
Data = DataBuf 'initialize Variant Buffer...
...
rc = Upic1.PrepareToReceive()
...
rc = Upic1.ReceiveMsg(Data, 0, Len(Data), length)
```
**Restart - initiate restart**

Calling this method initiates a restart, depending on the UTM generation in the partner system (UTM user ID must have been generated with RESTART=YES).

In the case of a service restart, openUTM restores the service context user-specifically. Therefore when *Restart* is called the relevant UTM user ID must be set in the *UserID* property and - if the UTM user ID is protected by a password - this password must be specified in the *Password* property.

This method can only be called at the beginning of a program: *SendMsg* must not be called beforehand.

*Restart* establishes a conversation with the program unit name KDCDISP and sends an empty message. If the call was successful, the method returns with the value TRUE: processing and communication with the UTM service can be resumed. The program will therefore call the *RequestForData* or *ReceiveMsg* method.

**Parameters**

This method has no parameters.

**Return value**

Type: BOOL

This method returns the value TRUE if a restart was initiated successfully. If FALSE is returned, further information can be obtained via the *GetLastError* method.

**Example in Visual Basic**

```vbnet
Dim DataBuf As String *100
Dim Data As Variant
Dim length As Long
Dim rc As Boolean
...
Data = DataBuf 'initialize Variant Buffer...
...
rc = Upic1.Restart()
...
rc = Upic1.ReceiveMsg(Data, 0, Len(Data), length)
```
**Disconnect - sign off from UPIC carrier system**

Calling this method signs the program off from the UPIC carrier system. A new conversation can only be established with the `SendMsg` or `Restart` methods.

**Parameters**

This method has no parameters.

**Return value**

Type: BOOL

This method returns the value TRUE if the program was successfully signed off from the UPIC carrier system. If FALSE is returned, further information can be obtained via the `GetLastError` method.

**Example in Visual Basic**

```vbnet
Dim rc As Boolean
...
rc = Upic1.Disconnect()
```

**InConversation - check that the conversation is still running**

The `InConversation` method indicates whether the conversation with the UTM service is still running. This method is important for establishing whether the conversation is still active after a successful `ReceiveMsg` call.

**Parameters**

This method has no parameters.

**Return value**

Type: BOOL

This function returns the value TRUE if a conversation currently exists. Otherwise, it returns the value FALSE. A new conversation can then be established with the `SendMsg` method (or, if necessary, `Restart`).

**Example in Visual Basic**

```vbnet
Dim rc As Boolean
...
rc = Upic1.InConversation()
```
InSendingState - query right to SendMsg

The InSendingState method indicates whether the program may call the SendMsg method.

The right to call the SendMsg method is not to be confused with permission to send as defined in UPIC. The right to call the SendMsg method includes the UPIC states: Start, Reset, Initialize and Send.

Parameters

This method has no parameters.

Return value

Type: BOOL

This method returns the value TRUE if the program may call the SendMsg method. Otherwise it returns the value FALSE, in which case the program has the right to receive.

Example in Visual Basic

Dim Data As String
Dim rc As Boolean

... rc = Upic1.InSendingState()
If rc = True Then
  rc = Upic1.SendMsg(Data, 0, Len(Data))
End If
**IncompleteDataReceived - check completeness of received data**

The `IncompleteDataReceived` method is used to establish whether a `ReceiveMsg` call must be followed by further receive calls in order to retrieve all data from the UTM partner.

**Parameters**

This method has no parameters.

**Return value**

Type: `BOOL`

This method returns the value `TRUE` if there is still receive data to be retrieved. Otherwise, it returns the value `FALSE`. The value always refers to the last `ReceiveMsg` call.

**Example in Visual Basic**

```vbnet
Dim DataBuf As String *100
Dim Data As Variant
Dim length As Long
Dim rc As Boolean
...
Data = DataBuf 'initialize Variant Buffer...
...
Do
...
rc = Upic1.ReceiveMsg(Data, 0, Len(Data), length)
...
Loop While Upic1.IncompleteDataReceived()
```
GetLastError - query UPIC error code and error information

The GetLastError method returns the UPIC error code of the method which produced the latest error. In addition, further information is passed on via the first parameter. If only makes sense to call this method directly after a failed method call.

Parameters

1st parameter: Pointer to a unicode string (basic string) which records the error text.

C++ notation: BSTR *Text

Return value

Type: long

This method returns a UPIC error code (see the header file UPIC.H).

Example in Visual Basic

Dim rcUpic As Long
Dim upicErrorStr As String
...
rcUpic = Upic1.GetLastError(upicErrorStr)
ConvertIncoming - convert EBCDIC to local code

The `ConvertIncoming` method converts data from EBCDIC to the code used on the local system.

Parameters

1st parameter: Unstructured data or basic strings which are to be converted.

C++ notation: `const VARIANT FAR* Data`

where:

- `Data.vt` equals `(VT_ARRAY | VT_UI1)` and the dimension of `Data` equals 1 or
- `Data.vt` equals VT_BSTR.

2nd parameter: Index (beginning with 0) as of which conversion takes place.

C++ notation: `long StartAt`

3rd Parameter: Length of the data to be converted.

C++ notation: `long ConvertLength`

Return value

Type: BOOL

This method returns the value TRUE after successful conversion. If FALSE is returned, further information can be obtained via the `GetLastError` method.

Example in Visual Basic

```vbnet
dim Data as Variant
dim rc as Boolean
...
rc = Upic1.ConvertIncoming(Data, 0, Len(Data))```
**ConvertOutgoing - convert local code to EBCDIC**

The `ConvertOutgoing` method converts data from the code used on the local system to EBCDIC.

**Parameters**

1st parameter: Unstructured data or basic strings which are to be converted.

C++ notation: `const VARIANT FAR* Data`

where:

- `Data.vt` equals `(VT_ARRAY | VT_UI1)` and the dimension of `Data` equals 1 or
- `Data.vt` equals `VT_BSTR`.

2nd parameter: Index (beginning with 0) as of which conversion takes place.

C++ notation: `long StartAt`

3rd parameter: Length of the data to be converted.

C++ notation: `long ConvertLength`

**Return value**

Type: `BOOL`

This method returns the value TRUE after successful conversion. If FALSE is returned, further information can be obtained via the `GetLastError` method.

**Example in Visual Basic**

```vbnet
Dim Data As Variant
Dim rc As Boolean
...
rc = Upic1.ConvertOutgoing(Data, 0, Len(Data))
```
3.2 **DataReceived event**

*UpicB.ocx* provides the *DataReceived* event.

This is only initiated when information on a non-blocking receive service has arrived. This can be, for example, received data or error codes.

A program starts a non-blocking receive service by calling the *RequestForData* method. This method returns immediately and the program can execute other activities. The program is informed of the arrival of information by the *DataReceived* event. It can then fetch the data or the error code with a *ReceiveMsg* call.

In the time from the successful call of the *RequestForData* method until the data is retrieved with the *ReceiveMsg* method, no other *UpicB.ocx* method may be called and no other *UpicB.ocx* property may be modified.

**Internal sequence**

The *RequestForData* method starts a blocking *Receive/Receive_Mapped_Data* call of the UPIC interface internally via a further thread and returns immediately.

When the internally triggered blocking *Receive/Receive_Mapped_Data* call returns, e.g. when the receive data has arrived, an error has occurred during receiving or the receive timer has expired, the *DataReceived* event is delivered to the client program.
3.3 Property Pages window

The Property Pages window is used for easy entry of default values for the properties. This is particularly useful for the FunctionKey property, which has the type long. Here a known name is assigned to a numerical value. The meaning of the individual properties is described in section “Properties of UpicB.ocx” on page 41.

The following screenshots show the individual tab sheets of the property pages.
**ActiveX control UpicB.ocx**

Property Pages window

---

**Property Pages**

- **General**
- **Function Key**
- **Receive**
- **Trace**

**Receive**

- **Timer:**
  - [ ] Blocking receive

**Location:**

[ ] Browse...

---

**Property Pages**

- **General**
- **Function Key**
- **Receive**
- **Trace**

---
3.4 Application example

You can try out this example when you have integrated `UpicB.ocx` into a Visual Basic form (see page 38), added two command buttons and a list box to the form and placed the sample code in the associated code window.

Such a form could look like this:

![Form with command buttons and list box](image)

When you run the example, clicking on **Command1** sends data to a UTM service and the result is received with `ReceiveMsg`.

Clicking on **Command2** starts a non-blocking receive service after the data is sent. Execution of the `Upic1_DataReceived()` routine is triggered by the “DataReceived” event.

Each call and its outcome are logged in the list box.

A more complex Visual Basic example can be found in the installation directory under: `upic-dir\sample\UpicCOM\OCXTestContainer`
Sample code

In the following example, some of the statements have been spread over two lines so that they will fit on the page. To avoid any error reports from Visual Basic, each of these statements must take up one line only.

' Global Variable for Data
Dim Data As String

' Call for Disconnect method
Private Sub DisconnectUpic1()
    Dim rc As Boolean
    rc = Upic1.Disconnect()
    If (rc = False) Then
        upicError("Disconnect")
    Else
        ' Log call in list1
        List1.AddItem "Disconnect"
    End If
End Sub

' Call for ReceiveMsg method
Private Function ReceiveUpic1() As String
    Dim ReceivedDataLength As Long
    Dim rcvData As String
    rcvData = String(10, " ")
    rc = Upic1.ReceiveMsg(rcvData, 0, 10, ReceivedDataLength)
    If (rc = False) Then
        upicError("ReceiveMsg")
        DisconnectUpic1
    Else
        List1.AddItem "ReceiveMsg (" +rcvData+ ")"
        ReceiveUpic1 = Left(rcvData, ReceivedDataLength)
    End If
End Function

' Output of last error
Private Sub upicError(lastFunc As String)
    Dim rcUpic As Long
    Dim upicErrorString As String
    rcUpic = Upic1.GetLastError(upicErrorString)
    List1.AddItem "Error": Str(rcUpic) + ": in Methode ": + lastFunc + Upic1
End Sub
Private Sub Command1_Click()
    Dim rc As Boolean
    'Set properties
    Upic1.LocalName = "PUPL1000"
    Upic1.SymDestName = "CSPIEGXX"
    Upic1.TPName = "CSPIEG"
    Data = "1.Hello.world!"
    'send data
    rc = Upic1.SendMsg(Data, 0, Len(Data))
    If (rc = False) Then
        upicError ("SendMsg")
        DisconnectUpic1
        Exit Sub
    Else
        'Log call in list1
        List1.AddItem "SendMsg (" + Data + ")"
        'receive complete data
        Data = ""
        Do
            Data = Data + ReceiveUpic1
        Loop While (Upic1.IncompleteDataReceived And Upic1.InConversation
            And Not Upic1.InSendingState)
        End If
        List1.AddItem "complete Data received (" + Data + ")"
    End If
End Sub
Private Sub Command2_Click()
    Dim rc As Boolean
    'Set properties
    Upic1.LocalName = "PUPL1000"
    Upic1.SymDestName = "CSPIEGXX"
    Upic1.TPName = "CSPIEG"
    
    Data = "2.Hello.world!"
    'send data
    rc = Upic1.SendMsg(Data, 0, Len(Data))
    If (rc = False) Then
        upicError ("SendMsg")
        DisconnectUpic1
        Exit Sub
    Else
        'Log call in list1
        List1.AddItem "SendMsg (" + Data + ")"
        Data = ""
        'Start non-blocking receive request
        rc = Upic1.RequestForData(10)
        If (rc = False) Then
            upicError ("RequestForData")
            DisconnectUpic1
        Else
            List1.AddItem "RequestForData"
        End If
    End If
End Sub

Private Sub Upic1_DataReceived()
    'receive Data
    Data = Data + ReceiveUpic1
    If (Upic1.IncompleteDataReceived Or Upic1.InConversation And Not Upic1.InSendingState) Then
        'non-blocking receive request
        rc = Upic1.RequestForData(10)
        If (rc = False) Then
            upicError ("RequestForData")
            DisconnectUpic1
        Else
            List1.AddItem "RequestForData"
        End If
    Else
        List1.AddItem "complete Data received (" + Data + ")"
    End If
End Sub
3.5 The ActiveX automation server UpicAutSvr.exe

As well as UpicB.ocx, you will receive with openUTM-Client a fully functional external ActiveX server, the UPIC automation server UpicAutSvr.exe.

With this automation module, the services of UpicB.ocx can also be used within standard Windows applications such as Microsoft Office. For example, for standard Word letters or for Excel diagrams you can implement automatic access to data provided by UTM server applications.

The ActiveX automation server is automatically entered in the system’s Registry under the name Upic.UpicAutSvr when openUTM-Client is installed.

The UPIC ActiveX automation server is implemented in Visual Basic V6.0. It is provided both as an executable program (.exe file) and in source form, so that you can use it as a model for programming your own automation modules. UpicAutSvr.exe is therefore not designed as a fixed component of the product, but is intended as an example and is therefore not subject to maintenance obligations.

The executable file is located in:
<installation_directory>\sys\UpicAutSvr.exe

The accompanying source is located in:
<installation_directory>\sample\UpicCOM\UpicAutSvr
4  C++ class CUpic

This chapter contains information about:

– Helper classes
– CUpic class

However, an introduction to configuring UPIC is presented first.

4.1 Introduction

The CUpic class is a wrapper class for the openUTM-Client interface. UPIC must be configured correctly before this class can be used.

There are two configuration options:

– Configuration using helper classes CUpicLocAddr and CUpicRemAddr
– External configuration using a side information file (upicfile)

4.1.1 Configuration using Helper classes CUpicLocAddr and CUpicRemAddr

The simplest case involves using the constructors, e.g.

\[
\text{CUpicLocAddr("upicw32")}
\]

and

\[
\text{CUpicRemAddr("", "sample", "local", 30000)}
\]
4.1.2 Configuration using a side information file (upicfile)

A `upicfile` must be available for external configuration. This file must at least contain the default entries for a local name and for the `Symbolic_Destination_Name`, e.g.:

```
LN.DEFAULT upicw32;
SD.DEFAULT sample.local hello PORT=30000;
```

For a description of configuring the `upicfile`, see section “The side information file (upicfile)” on page 323.

4.1.3 The CUpic class on thread-capable systems

A worker thread is created for each CUpic object on thread-capable systems (see page 32). In this way, several UPIC conversations can be active in parallel in an application. The threads are created and controlled transparently by the CUpic class.
4.2 Helper classes

Helper classes define address and security objects. The address objects can be passed to the CUpic objects as arguments of the constructors. Security objects can only be set using the property handler function SetSecurity()

4.2.1 CUpicLocAddr

CUpicLocAddr defines a local UPIC address.

4.2.1.1 Constructors

CUpicLocAddr()

The DEFAULT name if the local name in the side information is used.

CUpicLocAddr(const char * local_name)

local_name is passed to the Enable_UTM_UPIC call as an argument.

CUpicLocAddr(const char * local_name , const char * tsel_name , CM_INT32 port)

The local RFC1006 address is defined explicitly.

local_name is passed to the Enable_UTM_UPIC call as an argument. The value ZERO means that an empty local_name is used.

tsel_name is used directly as a local name and passed to the Specify_Local_Tsel() call.
If tsel_name only contains uppercase letters and digits, then the TRANSDATA Tsel_Format is used, otherwise EBCDIC.

port is passed to the Enable_UTM_UPIC call as an argument.
### 4.2.1.2 Member functions

#### void SetTselName (const char * name)

tsel\_name is used directly as a local name and passed to the *Specify\_Local\_Tsel()* call. If tsel\_name only contains uppercase letters and digits, then the TRANSDATA *Tsel\_Format* is used, otherwise EBCDIC.

#### void SetPort (CM\_INT32 port)

port is used as the local port number and passed to the *Specify\_Local\_Port()* call.

#### int SetTselFormat (const unsigned char format)

The *Tsel\_Format* can be set with this function:
- 'A' for ASCII
- 'E' for EBCDIC
- 'T' for TRANSDATA
4.2.2  CUpicRemAddr

CUpicRemAddr defines a remote UPIC address.

4.2.2.1  Constructors

CUpicRemAddr()
The DEFAULT name for the remote address is used.

CUpicRemAddr (const char * sym_dest_name)
sym_dest_name is passed to the Initialize_Conversation call.

CUpicRemAddr( const char * sym_dest_name,
               const char * tsel_name,
               const char * host_name,
               CM_INT32 port)
The remote RFC1006 address is defined explicitly.

sym_dest_name is passed to the Initialize_Conversation call.
tsel_name is used directly as a remote name and passed to the Set_Partner_Tsel() call. If tsel_name only contains uppercase letters and digits, then the TRANSDATA Tsel_Format is used, otherwise EBCDIC.

host_name is used directly as a remote host address. Depending on the string format used, it is passed to the Set_Partner_Host_Name() or Set_Partner_IP_Address() call.

port is used as a remote port number and passed to the Set_Partner_Port() call.
4.2.2.2 Member functions

void SetTselName (const char * tsel_name)

    tsel_name is used directly as a remote name and passed to the
    Set_Partner_Tsel() call.
    If tsel_name only contains uppercase letters and digits, then the
    TRANSDATA Tsel_Format is used, otherwise EBCDIC.

void SetHost (const char * host)

    host is used directly as a remote host address. Depending on the string
    format used, it is passed to the Set_Partner_Host_Name() or
    Set_Partner_IP_Address() call.

void SetPort (CM_INT32 port)

    port is used as a remote port number and passed to the
    Set_Partner_Port() call.

int SetTselFormat (const unsigned char format)

The Tsel_Format can be set with this function:
    – 'A' for ASCII
    – 'E' for EBCDIC
    – 'T' for TRANSDATA
4.2.3 **CUpic security**

CUpic Security defines the security attributes for UPIC.

**CUpic Security ()**

No security is used.

**CUpic Security (char * uid)**

uid is passed to the `Set_Conversation_Security_User_ID()` call.

**CUpic Security (char * uid, char * pwd)**

uid is passed to the `Set_Conversation_Security_User_ID()` call.

pwd is passed to the `Set_Conversation_Password()` call.
4.3 ClassCUpic

A CUpic object represents a conversation with an openUTM service.

4.3.1 Constructors

CUpic()
The DEFAULT name of the local and remote address is used.

CUpic (CUpicLocAddr 1)
The specified local address and the DEFAULT name of the remote address are used.

CUpic (CUpicRemAddr 2)
The specified remote address and the DEFAULT name of the local address are used.

CUpic (CUpicLocAddr 1, CUpicRemAddr 2)
The specified local and remote addresses are used.

4.3.2 Property handlers

void SetLocal(CUpicLocAddr l)
Defines a new local address.

void SetRemote(CUpicRemAddr r)
Defines a new remote address.

void SetSecurity(CUpicSecurity s)
Defines new security attributes.

void SetEncryption(BOOL)
Activates encryption.
void SetFunctionKey(CM_FUNCTION_KEY)
Activates the function key on the next send.

void SetTPName(const char * name)
Sets the transaction code (TAC) for a new conversation.
The function expects a string with the length strlen(name).

void SetMapName(const char * name)
Sets the map name for subsequent send calls.
The function expects a string with the length strlen(name).

void GetTPName(char * name)
Reads the currently valid transaction code (TAC).
The function copies a string with a following `\0` to the specified address. The target string must at least be declared as char name [9].

void GetMapName(char * name)
Reads out the last map name received.
The function copies a string with a following `\0` to the specified address. The target string must at least be declared as char name [9].
4.3.3 Function calls

int Snd (  
    const void * snd_buffer  
    , CM_INT32 send_len  
)

Sends the specified data. If no conversation is active, then all calls required to do so are 
implemented implicitly.

Result:
CUPIC_OK The call was successful.
CUPIC_ERROR An error has occurred. The GetLastError() call can be used for 
obtaining more information.

int SndLast (  
    const void * snd_buffer  
    , CM_INT32 send_len  
)

Sends the specified data and returns the send authorization. If no conversation is active, 
then all calls required to activate a conversation are made implicitly.

Result:
CUPIC_OK The call was successful.
CUPIC_ERROR An error has occurred. The GetLastError() call can be used for 
obtaining more information.

int Rcv (  
    void * rcv_buffer  
    , CM_INT32 buflen  
    , CM_INT32 * rcv_len  
)

Receives a response.

Result:
CUPIC_OK The call was successful and the conversation is closed.
CUPIC_MORE_DATA The call was successful, but only part of the message was received. 
The value of rcv_buffer was too small for the complete message. 
Rcv() must be called again in order to obtain the remaining data.
C++ class CUpic

The call was successful and the message was read in completely. Further messages can be received. \texttt{Rcv()} must be called again to receive the next message.

The call was successful, the last complete message was read in and the conversation is still open. \texttt{Snd()}, \texttt{SndLast()}, \texttt{SndRcv()} or \texttt{Call()} must be called in order to send the next data.

An error has occurred. The \texttt{GetLastError()} call can be used for obtaining more information.

\begin{verbatim}
int RcvMulti ( 
    void * rcv_buffer 
    , CM_INT32 buflen 
    , CM_INT32 * rcv_len 
)
\end{verbatim}

This function makes it possible to receive several CPI-C messages. Several CPI-C messages with the same map name are grouped into a single message. This is particularly useful whenever several linemode messages are to be received as a single message.

Result:

The call was successful and the conversation is closed.

The call was successful, but only part of the message was received. The value of \texttt{rcv_buffer} was too small for the complete message. \texttt{Rcv()} must be called again in order to obtain the remaining data.

The call was successful and a message with a map name was read in completely. Further messages can be received. \texttt{Rcv()} must be called again to receive the next message.

The call was successful, the last complete message was read in and the conversation is still open. \texttt{Snd()}, \texttt{SndLast()}, \texttt{SndRcv()} or \texttt{Call()} must be called in order to send the next data.

An error has occurred. The \texttt{GetLastError()} call can be used for obtaining more information.
int SndRcv (
    const void * send_buffer
  , CM_INT32 send_len
  , void * rcv_buffer
  , CM_INT32 rcvbuf_len
  , CM_INT32 * rcv_len
)

Sends the specified data and receives at least one response. If no conversation is active, then all calls required to activate a conversation are made implicitly. This call is a combination of Snd () and Rcv ().

Result:

CUPIC_OK
The call was successful and the conversation is closed.

CUPIC_MORE_DATA
The call was successful, but only part of the message was received. The value of rcv_buffer was too small for the complete message. Rcv () must be called again in order to obtain the remaining data.

CUPIC_MORE_MSGS
The call was successful and the message was read in completely. Further messages can be received. Rcv () must be called again to receive the next message.

CUPIC_CONV_IS_OPEN
The call was successful, the last complete message was read in and the conversation is still open. Snd (), SndLast (), SndRcv () or Call () must be called in order to send the next data.

CUPIC_ERROR
An error has occurred. The GetLastError() call can be used for obtaining more information.

int Call (
    const void * send_buffer
  , CM_INT32 send_len
  , void * rcv_buffer
  , CM_INT32 rcvbuf_len
  , CM_INT32 * rcv_len
)

Sends the specified data and receives at least one response. If no conversation is active, then all calls required to activate a conversation are made implicitly. This call is a combination of Snd () and RcvMulti ().
C++ class CUpic

Result:

CUPIC_OK
The call was successful and the conversation is closed.

CUPIC_MORE_DATA
The call was successful, but only part of the message was received. The value of rcv_buffer was too small for the complete message. 
Rcv() must be called again in order to obtain the remaining data.

CUPIC_MORE_MSGS
The call was successful and a message with a map name was read in completely. Further messages can be received. Rcv() must be called again to receive the next message.

CUPIC_CONV_IS_OPEN
The call was successful, the last complete message was read in and the conversation is still open. Snd(), SndLast(), SndRcv() or Call() must be called in order to send the next data.

CUPIC_ERROR
An error has occurred. The GetLastError() call can be used for obtaining more information.

int Restart (
    void * rcv_buffer,
    CM_INT32 rcvbuf_len,
    CM_INT32 * rcv_len
)

Activates the restart of a previous conversation and receives the data with the call RcvMulti().

Result:

CUPIC_OK
The call was successful and the conversation is closed.

CUPIC_MORE_DATA
The call was successful, but only part of the message was received. The value of rcv_buffer was too small for the complete message.
Rcv() must be called again in order to obtain the remaining data.

CUPIC_MORE_MSGS
The call was successful and a message with a map name was read in completely. Further messages can be received. Rcv() must be called again to receive the next message.

CUPIC_CONV_IS_OPEN
The call was successful, the last complete message was read in and the conversation is still open. Snd(), SndLast(), SndRcv() or Call() must be called in order to send the next data.

CUPIC_ERROR
An error has occurred. The GetLastError() call can be used for obtaining more information.
void Reset()
Closes the active conversation and shuts down the transport connection.

BOOL Peek()
Tests whether data is ready to be received.

4.3.4 Public diagnostic function

char * GetLastError ()
Returns a text string which explains the error in more detail. Reset () has already been called if CUPIC_ERROR is returned for the function.

void GetLastError (const char ** error_text, CM_CALL_ID * c, CM_RETCODE * rc)
Returns a text string which explains the error in more detail. The last call (defined as CM_CALL_ID in upic.h) and the last UPIC return code are returned.

char * GetDiagContext ()
The CUpic class writes all its actions into a diagnostic context in printable format. This method supplies this information by returning a pointer to the corresponding area.

void ResetDiagContext ()
The CUpic class writes all its actions into a diagnostic context in printable format. This method resets the content of the diagnostic context.
4.4 Example

```c
#include "CUpic.h"
void main ( int argc, char * argv[] )
{
    char sbuf[1000];
    char rbuf[100000];
    CM_INT32 rcv_len;
    int rc;

    CUpic u;
    // Make a simple call based on configuration defaults memset (sbuf, '\0', sizeof(sbuf));
    rc = u.Call (sbuf, strlen(sbuf), rbuf, sizeof(rbuf), &rcv_len);

    if ( rc == CUPIC_OK )
    {
        printf ("%.s", rcv_len, rbuf);
    }
    else
    {
        printf ("%s", u.GetLastError());
    }

    // Make a simple admin call overwriting configuration defaults
    CUpicLocAddr 1 = CUpicLocAddr("its-me!", 4711);
    CUpicRemAddr r = CUpicRemAddr("sample", "127.0.0.1", 30000);
    CUpicSecurity s = CUpicSecurity("admin");

    u.SetLocal (1);
    u.SetRemote (r);
    u.SetSecurity (s);
    u.SetTPName ("KDCINF");

    strcpy (sbuf, "STAT");

    rc = u.Call (sbuf, strlen(sbuf), rbuf, sizeof(rbuf), &rcv_len);

    if ( rc == CUPIC_OK )
    {
        printf ("%.s", rcv_len, rbuf);
    }
    else
    {
        printf ("%s", u.GetLastError());
    }
}
```
Example

C++ class CUpic
5 CPI-C interface

With UPIC as the carrier system you can link CPI-C applications which run on your local system with UTM applications which run on Windows systems, Unix systems or BS2000/OSD. The UTM service requested by the client can use either the CPI-C or the KDCS interface of openUTM.

This chapter describes:

- the general structure of CPI-C client programs
- the exchange of messages between client and server
- conversion of the exchanged data in heterogeneous links
- programming notes for communication with UTM single-step and multi-step services
- the encryption procedures
- programming client programs that are linked to several services in parallel (multiple conversations). Multiple conversations are only possible if the client is running on a system that supports multithreading.
- the security functions of openUTM, which can be used when UPIC client programs are connected.
- the CPI-C functions supported by the UPIC carrier system. The individual CPI-C function calls are described in full (the CPI-C Specification of X/Open is therefore not necessary).

First, however, we will explain some CPI-C terms which are used in the following chapters.
5.1 CPI-C terms

The terms ‘conversation’, ‘conversation characteristics’, and ‘side information’ exist in CPI-C.

- A **conversation** is a communication relationship processed by a CPI-C program in a UTM service.
- **Conversation characteristics** describe the current parameters and features of a conversation, see page 87.
- In connection with the UPIC carrier system, **side information** basically describes the addressing information required for a conversation. The addressing information necessary for a conversation is contained in the **side information file (upicfile)**.

Conversation state

The state of a conversation reflects the last action of this conversation or defines the next actions that are permitted.

When you write a program that uses CPI-C calls, you must ensure that the appropriate calls are always used in the CPI-C program and in the UTM program unit. In particular, only the partner with send authorization is permitted to send data.

With the UPIC carrier system, a conversation can have one of the following states:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>The program is not signed on to the UPIC carrier system.</td>
</tr>
<tr>
<td></td>
<td>(before the <code>Enable_UTM_UPIC</code> call or after the <code>Disable_UTM_UPIC</code> call).</td>
</tr>
<tr>
<td>Reset</td>
<td>No conversation is assigned to the conversation_ID.</td>
</tr>
<tr>
<td>Initialize</td>
<td>The <code>Initialize_Conversation</code> call was terminated successfully and a</td>
</tr>
<tr>
<td></td>
<td><code>conversation_ID</code> was assigned to the conversation.</td>
</tr>
<tr>
<td>Send</td>
<td>The program is authorized to send data in the conversation.</td>
</tr>
<tr>
<td>Receive</td>
<td>The program can receive information via the conversation.</td>
</tr>
</tbody>
</table>

Table 2: Conversation states

At the beginning, a conversation is in the “Reset” state and then enters various follow-up states, depending on the actual calls issued and the information received from the partner program.
The “Send” and “Receive” states have a special role to play. This role is described in section “Exchange of messages with a UTM service” on page 92. A table of states can be found in the appendix on page 386. Here you will find the state changes of a CPI-C conversation, depending on the CPI-C calls and their results.

UPIC monitors the current state of a conversation. If the synchronization of the two sides is violated by an illegal call, this error is displayed with the value CM_PROGRAM_STATE_CHECK as the result of the call.

The X/Open CPI-C Specification defines further states, but these do not apply to the UPIC carrier system.

**Conversation characteristics**

The conversation characteristics are managed in a control block together with the side information of a conversation. This section describes the characteristics relevant to CPI-C with the UPIC carrier system, as well as the values assigned to these characteristics in the Initialize_Conversation call. The X/OPEN interface CPI-C contains additional characteristics which are not listed here.

There are three types of conversation characteristics:

- those that are preset
- those that can be modified using CPI-C calls
- those that are UPIC specific

The following conversation characteristics are preset:

<table>
<thead>
<tr>
<th>Conversation characteristics</th>
<th>Initialization value for Initialize_Conversation</th>
</tr>
</thead>
<tbody>
<tr>
<td>conversation_type</td>
<td>CM_MAPPED_CONVERSATION</td>
</tr>
<tr>
<td>return_control</td>
<td>CM_WHEN_SESSION_ALLOCATED</td>
</tr>
<tr>
<td>send_type</td>
<td>CM_BUFFER_DATA</td>
</tr>
<tr>
<td>sync_level</td>
<td>CM_NONE</td>
</tr>
</tbody>
</table>

Table 3: Preset conversation characteristics
The following conversation characteristics can be modified using CPI-C calls:

<table>
<thead>
<tr>
<th>Conversation characteristics</th>
<th>Initialization value for Initialize_Conversation</th>
</tr>
</thead>
<tbody>
<tr>
<td>deallocate_type</td>
<td>CM_DEALLOCATE_SYNC_LEVEL</td>
</tr>
<tr>
<td>partner_LU_name</td>
<td>Value from side information, dependent on the symbolic destination name</td>
</tr>
<tr>
<td>partner_LU_name_length</td>
<td>Length of partner_LU_name</td>
</tr>
<tr>
<td>receive_type</td>
<td>CM_RECEIVE_AND_WAIT</td>
</tr>
<tr>
<td>security_new_password</td>
<td>Empty</td>
</tr>
<tr>
<td>security_new_password_length</td>
<td>0</td>
</tr>
<tr>
<td>security_password</td>
<td>Blank</td>
</tr>
<tr>
<td>security_password_length</td>
<td>0</td>
</tr>
<tr>
<td>security_type</td>
<td>CM_SECURITY_NONE</td>
</tr>
<tr>
<td>security_user_ID</td>
<td>Blank</td>
</tr>
<tr>
<td>security_user_ID_length</td>
<td>0</td>
</tr>
<tr>
<td>TP_name</td>
<td>Value from side information, dependent on the symbolic destination name</td>
</tr>
<tr>
<td>TP_name_length</td>
<td>Length of TP_name</td>
</tr>
</tbody>
</table>

Table 4: Conversation characteristics which can be modified
The following conversation characteristics are UPIC specific and can be modified. The distinction is made between characteristics for a partner application and values for a local application:

<table>
<thead>
<tr>
<th>Conversation characteristics</th>
<th>Initialization value for Initialize_Conversation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER_CONVERSION</td>
<td>CM_NO_CHARACTER_CONVERSION</td>
</tr>
<tr>
<td>Client_CONTEXT</td>
<td>empty</td>
</tr>
<tr>
<td>COMMUNICATION_PROTOCOL</td>
<td>CM_COMMUNICATION_PROTOCOL_40</td>
</tr>
<tr>
<td>ENCRYPTION-LEVEL</td>
<td>0</td>
</tr>
<tr>
<td>PORT</td>
<td>102</td>
</tr>
<tr>
<td>T-SEL</td>
<td>Value derived from partner_LU_name</td>
</tr>
<tr>
<td>T-SEL-FORMAT</td>
<td>Value derived from partner_LU_name</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>Value derived from partner_LU_name</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>Not initialized</td>
</tr>
<tr>
<td>RSA-KEY</td>
<td>Allocated by the UTM application</td>
</tr>
<tr>
<td>SECONDARY_RETURN_CODE</td>
<td>CM_RETURN_TYPE_SECONDARY</td>
</tr>
<tr>
<td>TRANSACTION_STATE</td>
<td>empty</td>
</tr>
</tbody>
</table>

Table 5: UPIC specific conversation characteristics for remote applications

<table>
<thead>
<tr>
<th>Values for local applications</th>
<th>Initialization value for Enable_UTM_UPIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT</td>
<td>102</td>
</tr>
<tr>
<td>T-SEL</td>
<td>Value derived from local application name</td>
</tr>
<tr>
<td>T-SEL-FORMAT</td>
<td>Value derived from local application name</td>
</tr>
</tbody>
</table>

Table 6: UPIC specific values for local applications

The characteristics and local values are not explained in greater detail. This list is merely given to enable the conversation characteristics in the CPI-C interface provided by UPIC to be compared with those in the X/Open CPI-C interface. A detailed explanation can be found in the X/Open specification “CPI-C Specification Version 2".
Side information

Because the addressing information is dependent on the respective configuration, CPI-C applications use the following symbolic names for addressing.

- **Symbolic Destination Name**
  The *Symbolic Destination Name* addresses the communication partner. The Symbolic Destination Name comprises two components:

  - **partner LU name**
    addresses the partner UTM application and can be overwritten in the program by `Set_Partner_LU-name`.

  - **TP name**
    addresses the UTM service within the UTM partner application. *TP name* is a transaction code and can be overwritten by the program with `Set_TP_Name`, e.g. `TP_name=KDCDISP` for the restart.

    The UTM service addressed by this transaction code is started as soon as the program has issued the first `Receive` call or a `Prepare_To_Receive` call.

  - **Keywords**
    further UPIC-specific conversation characteristics can be set with various keywords. A program can overwrite these characteristics with the corresponding CPI-C calls (for example, `Set_Encryption_Level`).

    The *Symbolic Destination Name* is linked with the “real” addressing (*partner LU_Name*, *TP_Name*) using the `upicfile`. *partner LU_name*, *TP_Name* and the keywords are just some of the conversation characteristics described below.

- **local name**
  The *local name* assigns the local application name for the local application. A symbolic name can be assigned for the *local name* in the `upicfile`. UPIC-local values can be set using keywords. This means that the name assigned by the program is independent of the name used in the TNS or UTM generation. A program can overwrite these characteristics with the corresponding CPI-C calls (for example, `Specify_Local_Tsel`).

A description of how the `upicfile` is created and how the entries are linked with the TNS and UTM generation can be found in section “Coordination with the partner configuration” on page 343.

When a `upicfile` is used, this offers the advantage that TNS and UTM generation can be modified (e.g. by moving the UTM server application to another system) without the client programs having to be modified.
5.2 General structure of a CPI-C application

A CPI-C application is a main program which generally includes the following:

- operation of an interface to a presentation system
- internal processing routines (operation of other interfaces if necessary)
- operation of the CPI-C interface (to a UTM application)
- overview of special CPI-C and UTM functions which the clients can use via UPIC

Sequence of calls in a CPI-C application

The following rules apply to the interface calls described in section “CPI-C calls in UPIC” on page 132:

1. The first CPI-C function call in your program must be Enable_UTM_UPIC and the last call must be Disable_UTM_UPIC. Between these two calls, you can repeat the other CPI-C calls as often as desired in accordance with the rules described below. Enable_UTM_UPIC starts the runtime environment for the client.

2. After calling Enable_UTM_UPIC, you can use the Specif_... calls to modify the UPIC-specific values of the local application.

3. You must initialize the conversation characteristics with Initialize_Conversation. The characteristics are described on page 87.

4. After initialization you can set or modify various conversation characteristics using the Set_... calls (see the modifiable characteristics on page 88).

5. You must establish the conversation with the Allocate call.

6. Following an Allocate call you can perform processing with the calls Send_Data, Send_Mapped_Data as well as Prepare_To_Receive, Receive and Receive_Mapped_Data. After the Allocate call, however, a Send_Data or Send_Mapped_Data call has to be made first before the program can receive data from the UTM server with Receive or Receive_Mapped_Data. For more information on the Send and Receive calls, see section “Exchange of messages with a UTM service” on page 92.

If a CPI-C program is to hold several conversations consecutively, for performance reasons it is advisable to issue only one Enable_UTM_UPIC and one Disable_UTM_UPIC call in a CPI-C application, i.e. you should not issue an Enable call before each Initialize_Conversation and a Disable call each time the conversation is terminated.

If a CPI-C program is to hold several conversations simultaneously, and Enable_UTM_UPIC call must be made for each of these conversations before the Initialize_Conversation. All CPI-C calls belonging to a conversation must occur in the same thread. See section “Multiple conversations” on page 125.
Communication with openUTM < V4.0

When a CPI-C program communicates with a UTM application based on openUTM Version 4.0 or higher, you must set the COMMUNICATION_PROTOCOL characteristic accordingly. Use the CPI-C Set_Communication_Protocol call to do this (see page 218) or make an entry in the upicfile (see page 328).

Note that there are some functions that you cannot use for communication with openUTM < V4.0. Corresponding CPI-C calls are rejected with one of the following return codes: CM_MAP_ROUTINE_ERROR, CM_ENCRYPTION_NOT_SUPPORTED or CM_CALL_NOT_SUPPORTED.

5.3 Exchange of messages with a UTM service

Once a conversation has been established between a client and a UTM service, the client must pass messages to the UTM service to control it. The service sends the client the processing result in the form of a message. Note, however, that only one side (client or service) at a time may send data in a conversation. We say that this side of the conversation has “permission to send”. Permission to send must be explicitly transferred to the other side of the conversation so that the partner can send data.

This section describes
- how the exchange of messages works,
- what you have to consider when programming a client application and
- which functions are available for the exchange of messages.

In section “Communicating with the UTM server” on page 107 you will find detailed examples of communication between client and UTM server application, contrasting the program sequence on the client side and the server side (KDCS interface).
5.3.1 Sending a message and starting a UTM service

The following diagram illustrates the sequence in the client program via which the client starts the service in the UTM server application and transfers a message to the service.

Explanation of the diagram

1. Following the `Allocate` call, the conversation is “established” and a connection to the UTM server has been set up. The UTM service, however, is not yet started. UPIC now manages an internal buffer to which the data from the conversation is written.

2. Following the `Allocate` call, the client is in the “Send” state; it has permission to send data to the conversation and must now transfer a message for the addressed service (`TP_Name`) to UPIC. The message must contain the input data to be processed by the service. The following `Send` calls are available to the client for this:

   - `Send_Data`
   - `Send_Mapped_Data`
After the *Allocate* call you may still modify the conversation characteristic *receive_type* and the values for the receive timer and the function key using *Set_*...* calls.

*Send_Mapped_Data* differs from the *Send_Data* call in that, as well as the message, format names are also sent to the server. In the same way, the client can receive data together with the format names from the service with *Receive_Mapped_Data*. See section “Sending and receiving formats” on page 98.

The *Send* call writes the data from UPIC into a local send buffer which is uniquely assigned to the UTM service on the local system. The client can issue several *Send* calls for transferring the message.

If the UTM service does not need any data for processing the request, the client must send an empty message to the server.

3. Once the client has transferred the message completely to UPIC, it must pass on send authorization to the server by changing to the “Receive” state. The following CPI-C calls are available for this:

- *Receive*
- *Receive_Mapped_Data*
- *Prepare_To_Receive*

Only now does UPIC transfer the last section of the send buffer to the UTM service together with permission to send. The corresponding program unit of the UTM server application is started.

If you use a *Receive* call to transfer permission to send to the UTM application, the client transfers permission to send and then waits in the *Receive* for the response from the service (blocking receive; see section “Receiving a message, blocking and non-blocking receive” on page 95).

The *Prepare_To_Receive* call causes the local UPIC send buffer to be transferred immediately to the server together with permission to send. The client switches to the “Receive” state but does not receive any data yet. When the response is received from the UTM service, the client must call *Receive* or *Receive_Mapped_Data*. Before this *Receive* call, however, the client cannot execute further (local) processing steps which do not use the CPI-C interface. Because the conversation is in the “Receive” state, only the CPI-C calls *Set_Receive_Type*, *Set_Receive_Timer*, and *Set_Function_Key* are allowed between *Prepare_To_Receive* and the *Receive* or *Receive_Mapped_Data* call.

*Prepare_To_Receive* is useful if you are starting a “long-running” service which will not necessarily produce a reply, e.g. services with several database accesses or with distributed transaction processing between the UTM partner application and other server applications. The client program and the process are then not blocked for the entire processing time.
5.3.2 Receiving a message, blocking and non-blocking receive

The UTM service transfers its results in the form of a message or several message segments to the client. This can also be an empty message. Moreover, the UTM server either transfers permission to send to the client or terminates the conversation. The message from the UTM service is received by UPIC and stored locally in a receive buffer. The client can pick up the message from the receive buffer as required using one of the following Receive calls:

Receive
Receive_Mapped_Data

Every message segment from the UTM service (every MPUT NT/NE) must be received with its own Receive call. If the status_received field is set to CM_SEND_RECEIVED for the Receive call, the client receives permission to send.

When the UTM service terminates (PEND FI), the conversation is terminated by the server. In the Receive, the return code CM_DEALLOCATE_NORMAL is returned to the client and the conversation switches to the “Reset” state.

A CPI-C program must always issue at least one Receive call, i.e. Send calls without a following Receive call are not permitted.
The following diagram shows how messages are received in the client program.

Figure 8: Client receives a message from server, conversation is shut down

**Explanation of the diagram**

1. With the `Set_Receive_Type` call you can specify whether the data is to be received with or without blocking. Whether a `Receive` call is processed with blocking or without depends on the value of the conversation characteristic `receive_type`. After initialization of the conversation characteristics with the `Initialize_Conversation` call, a blocking `Receive` is set for the conversation. You can change this default setting using the `Set_Receive_Type` call.

With a **blocking** `Receive` call (receive_type=CM_RECEIVE_AND_WAIT) the client program waits in the `Receive` or `Receive_Mapped_Data` until data from the server arrives for the conversation or the call is interrupted by a timer. Only then is control returned to the client program and the program run can be resumed.

If you are working with the blocking receive, you should make sure that the program does not wait “for ever” by setting appropriate timers in the UTM server application (see the openUTM manual “Administering Applications” and the openUTM manual "Generating Applications"). On the client side, a timeout timer can be set for the blocking `Receive` with `Set_Receive_Timer`. 
In the case of a **non-blocking** *Receive* call (*receive_type*=CM_RECEIVE_IMMEDIATE), control is returned to the program immediately. If data from the service is present at the time of the call, it is transferred to the program. If there is no data present at the time of the call, the call returns the return code CM_UNSUCCESSFUL.

The *receive_type* characteristic can be changed as often as you like within the conversation. For each *Receive*, the setting defined by the last *Set_Receive_Type* call before the *Receive* applies.

*Upic local:*
Local connection via UPIC local does not support the non-blocking *Receive* or the *Set_Receive_Type* call.

2. With the *Receive* or *Receive_Mapped_Data* call, the client reads the data from the receive buffer. If data is present, the *Receive* call passes the data directly to the client program. The remaining course of the client program depends on the result of the *Receive* call (fields *data_received*, *status_received*, *return_code*). The following results can occur:

- Once the program has fully read the message with the *Receive* call (*data_received*=CM_COMPLETE_DATA_RECEIVED) and the UTM service has terminated the conversation (PEND FI called), the program switches to the “Reset” state. It can now establish a new conversation or sign off from UPIC with *Disable_UTM_UPIC*.

- The program has not yet read all message segments that were received from the service. It must continue to issue *Receive* calls until *data_received* assumes the value CM_COMPLETE_DATA_RECEIVED. One *Receive* call must be issued for each message segment the service sends (MPUT NT).

- The program has read the full message from the service and the service transfers permission to send to the client (*status_received*=CM_SEND_RECEIVED). The next thing the client must do is issue at least one *Send* call and then issue *Receive* calls again. In this case the UTM service is a multi-step service (the program unit has terminated with PEND KP).

3. Once the last conversation has terminated, the client program calls *Disable_UTM_UPIC* in order to sign off from UPIC.
5.3.3 Sending and receiving formats

A CPI-C client using the UPIC carrier system can together with a user message, send format names to a UTM service and receive format names from a UTM service.

The format names transferred with the user message can be used to describe the data format of the user data. The user data and format names that are exchanged between client and server are transferred transparently, i.e. they can contain any bit combinations, which must be interpreted by the recipient of the message. The user message is not processed by a form generating system by means of the format name.

The format names exchanged between UPIC and UTM can generally be freely selected, as can the structure. The structure information is important if programs written for terminals are to be used to communicate with UPIC clients. In this event, the format ID plays a role. The format ID is made up of a prefix (-, +, # or *) and the actual format name.

UPIC clients and UTM programs use the format names which are defined in the UTM application in order to specify the structuring characteristics of a message. For each format ID that the UTM application recognizes there is a data structure (addressing aid) in the UTM application. A UPIC client can also use this function to call UTM applications which communicate with terminals using formats. To do this the client program must transfer the format ID that the UTM program expects. The user message is then made up according to the format IDs.

In the same way, when sending format data the UTM server application passes on to the client program the format identifier which describes the structure of the message area.
CPI-C calls for exchanging format data

Because the CPI-C interface does not have its own concept for transferring format names to the interface, UPIC uses the functions

Send_Mapped_Data and Receive_Mapped_Data

to send and receive messages together with format names.

To send format data to the UTM server application, call Send_Mapped_Data. In the map_name field of the call, the client transfers the format ID as structure information for the message which is to be sent to the UTM server application. The message must be structured according to the format defined in the server application. Send_Mapped_Data is described on section “Send_Mapped_Data - Sending data and format identifier” on page 210.

If the UTM service returns a format, the client program must call Receive_Mapped_Data in order to receive the message from the UTM service together with the format ID. In the map_name field, UPIC transfers the format ID used by the server to structure the message. In the client program the message must be interpreted according to the structuring used by the UTM service. Receive_Mapped_Data is described on section “Receive_Mapped_Data - Receiving data and format identifier from a UTM service” on page 196.

If several partial formats are to be sent to a UTM service, the client program must issue a separate Send_Mapped_Data call for each one. The UTM service reads each partial format with a separate MGET NT call.

By the same token, if a message from the UTM service consists of several partial formats, the client program must issue a Receive_Mapped_Data call for each partial format.
Detailed information on working with formats in a UTM server application can be found in the openUTM manual "Programming Applications with KDCS".
openUTM format identifiers and -format types

The format names exchanged between a UPIC client program and a UTM program unit can consist of up to 8 characters of your choice. The important thing to remember is that both the communication partners must agree on the structure and meaning of the user data transferred using the format name.

If a client program calls a UTM program unit that also communicates with terminals using format IDs, the format ID must correspond with the rules for form generating systems supported by openUTM. These format IDs consist of:

- a one-byte prefix specifying the type of the format (possible values are “#”, “+”, “#” and with openUTM on BS2000/OSD also “.”)
- a format name up to 7 characters long.

The format types can be classified as follows:

* formats:
The display attributes of the format fields cannot be modified by a UTM program unit. Only the contents of the data fields are transferred.

+ formats and # formats:
A UTM program unit can modify the display attributes of the data fields or global attributes. The data fields are therefore assigned attribute fields or blocks. If a +format or a # format is exchanged, the client program must take these attribute fields into account.

- formats
These are only possible in openUTM on BS2000/OSD applications. They are formats which are created with the FORMAT event exit.

For more about format IDs and types, see the openUTM manual "Programming Applications with KDCS".

The rules for format IDs do not need to be observed if a UTM program unit only communicates with UPIC-Client program units. Format generation systems do not play any part in this form of communication.
5.3.4 UTM function keys

In a UTM server application, function keys can be generated (F1, F2, ...F24 and in BS2000/OSD also K1 through K14). Each function key can be assigned via UTM generation a particular function, which openUTM executes when the function key is pressed.

A CPI-C client program can activate function keys in a UTM server application, provided that on the server side openUTM V4.0 or higher is being used, and the communication with the UTM server application takes place via the UPIC communication protocol 40.

For “pressing a UTM function key”, the function call Set_Function_Key is provided. Set_Function_Key is a UPIC-specific function which is not part of the functional scope of the X/Open-CPI-C interface.

With Set_Function_Key the client program specifies the function key which is to be activated in the UTM server application.

The return code assigned to this function key is transferred to the UTM service by openUTM at the first MGET call (KCRCCC field). The program-unit run of the UTM service can be controlled via the return code (e.g. a particular follow-up TAC can be started). To read the message from the client which sent it with Send_Mapped_Data, a second MGET call must be made.

Calling Set_Function_Key is only permitted in the “Send” and “Receive” states. The function key is transferred to the service together with the data of the following Send call.
CPI-C interface

Exchange of messages

Figure 10: Pressing a function key in a UTM server application

<table>
<thead>
<tr>
<th>CPI-C application</th>
<th>UTM application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable_UTM_UPIC</td>
<td>INIT</td>
</tr>
<tr>
<td>Initialize_Conversation</td>
<td>MGET</td>
</tr>
<tr>
<td>possibly</td>
<td>The call returns in KCRCCC the return</td>
</tr>
<tr>
<td>Set_TP_Name</td>
<td>code generated in SFUNC.</td>
</tr>
<tr>
<td>Set_...</td>
<td>MGET (message)</td>
</tr>
<tr>
<td>Allocate</td>
<td>...</td>
</tr>
<tr>
<td>Set_Function_Key()</td>
<td>MPUT (response)</td>
</tr>
<tr>
<td>Send_Mapped_Data (message)</td>
<td></td>
</tr>
<tr>
<td>possibly</td>
<td>PEND FI</td>
</tr>
<tr>
<td>Prepare_To_Receive</td>
<td></td>
</tr>
<tr>
<td>Receive</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Disable_UTM_UPIC</td>
<td></td>
</tr>
</tbody>
</table>

The call returns in KCRCCC the return code generated in SFUNC.
5.3.5 Cursor position

If, in a dialog step in a UTM program unit, a format output is intended and the cursor is to set to a field using the KDCSUR call, then this information will be transferred to UPIC. UTM uses the differences between the address of the specified field and the start address of the format to create an offset. This offset is transferred to the UPIC client and can be interrogated using the Extract_Cursor_Offset call.

The Extract_Cursor_Offset call delivers a return value. If this value is 0, KDCSCUR in the UTM program unit was not called, unless the cursor is to be set at the beginning of the format and the call really does result in the offset 0. If KDCSCUR is called in the UTM program unit, Extract_Cursor_Offset delivers the cursor address in the format, as a integer in a format relative to the start of the message area.

5.3.6 Code conversion

With a heterogeneous link to a UTM server application, it may be the case that different codes (ASCII, EBCDIC) are used in the client and the server systems, for example

- a client application running on a Unix system or a Windows system communicates with a UTM server application on a BS2000/OSD system.
- a client application running on a BS2000/OSD system communicates with a UTM server application on a Unix system or Windows system.

Unix systems and Windows systems use ASCII code, while BS2000/OSD systems use EBCDIC code. If an ASCII system is linked to an EBCDIC system, messages which contain printable characters (7 bit ASCII character set) can be converted, say for output. Pure binary data must not be converted. The conversion can take place either on the client side or on the server side. You must make sure that it only occurs once.

Code conversion for UPIC-Clients cannot be generated in openUTM (the MAP parameter for PTERM and TPOOL can only have the value USER for UPIC clients). Server-side conversion must therefore be carried out by the user in the program unit.

If the conversion is to take place in the client, two options are available with the UPIC carrier system:

- The CPI-C calls Convert_Incoming and Convert_Outgoing
  - In this case, the data is converted by the program. With Convert_Incoming you can convert a received message into the code used locally (see section “Convert_Incoming - Converting data from code of sender to local code” on page 139). With Convert_Outgoing you can convert the data to be sent (before it is sent) from the local code into the code of the recipient (see section “Convert_Outgoing - Converting data from local code to code of receiver” on page 140).
Automatic code conversion of the UPIC carrier system

You activate automatic code conversion for the connection to a specific server using the `CHARACTER_CONVERSION` conversation characteristic. You can activate `CHARACTER_CONVERSION` as follows:

- by entering a corresponding ID in the side information entry or the `upicfile` for this server (see section “Side information for standalone UTM applications” on page 324).

- or by means of the Set_Character_Conversion call.

When code conversion is activated, UPIC converts all data which arrives from this server into the locally used code before it is transferred to the client program, and all data sent from the client program to the server into the code of the server before it is sent. The client program no longer needs to deal with the conversion; `Convert_Incoming` and `Convert_Outgoing` must no longer be executed.

The automatic code conversion makes it possible with a single CPI-C program to communicate both with a UTM application on Unix systems or on Windows systems based on the ASCII code and with a UTM application on BS2000/OSD based on the EBCDIC code (if the user data does not contain any binary information that was falsified during the code conversion).

**CAUTION!**

You must remember that with a heterogeneous link the messages are converted only once. Only messages containing printable characters may be converted. No conversion at all is allowed with a homogeneous link and with the link Windows system <-> Unix system.

The Euro symbol has the value 0x80 in the Windows character set, 0xa4 in the ASCII character set and 0x9f in EBCDIC (general currency symbol). In the strictest sense these are not printable characters (8 bit ASCII character set).

In UPIC on Unix systems and UPIC on BS2000/OSD you can also modify the supplied conversion tables to suit your requirements, compile them and replace them in the UPIC library using the usual operation system methods.
5.3.7 User-defined code conversion for Windows systems

The conversion tables are located in a separate dynamic library. This means you can adapt the conversion tables to your own requirements. The files necessary to do this are installed in the `upic-dir\utmcnv32` directory. These are:

- `kcsaeaa.c`, C-source with the conversion tables of the previous UPIC versions. For reasons of compatibility, this file is contained in the `utmcnv32.dll` which is supplied with the product.

- `kcxaent.c`, C-source with complete conversion tables between the Windows character set and EBCDIC.

- `utmcnv32.def`, Def file with EXPORT statements.

- `utmcnv32.rc`, `resource.h`, Resource files with version information.

The version information is not absolutely essential in order to create the library.

Procedure with a Microsoft Visual C++ Developer Studio:

1. Create a new project called `utmcnv32` in the `upic-dir\utmcnv32` directory. The type of the project must be Dynamic Link Library.

2. Add the files `kcsaeaa.c` or `kcxaent.c`, `utmcnv32.def` and, if appropriate, `utmcnv32.rc` to the project.

3. Create `utmcnv32.dll` with this project.

4. Once the `utmcnv32.dll` library has been created successfully, you still have to copy it into the directory containing the UPIC library `upicw32.dll` or `upicws32.dll` which is loaded by your application. The directory is usually a subdirectory of the Windows system directory (e.g. `\Windows\System` or `Winnt\System32`).

5. Make certain that the original library `utmcnv32.dll` is either overwritten by copying or is deleted, otherwise it may be loaded inadvertently by the system instead of the new library.

The conversion tables are structured in the form of two character arrays with the size 256:

- `unsigned char kcsaebc[256]` for converting from ASCII characters to EBCDIC

- `unsigned char kcesasc[256]` for converting from EBCDIC characters to ASCII

The EBCDIC code of the ASCII character `n` is the value of the nth element of the character array `kcsaebc`, i.e. of `kcsaebc[n]`. 
Examples:

1. “M” has the ASCII code 4D hex or 77 dec. \texttt{kcsaebc[77]} is set to the value D4 hex, which is the EBCDIC code of “M”.

2. The Umlaut “Ä” (code C4 hex in ISO 8859-1, code 63 hex in EBCDIC.DF.04-1) is not included in the original conversion tables. If you want to integrate it into your own conversion tables, you must change the value of \texttt{kcsaebc[196]} from 0xff to 0xc4 (ASCII to EBCDIC conversion) and change the value of \texttt{kcseasc[99]} from 0x1a to 0xc4 (EBCDIC to ASCII conversion).

5.4 Communicating with the UTM server

In this section, examples are used to show how a CPI-C program can communicate with a UTM application in single-step and multi-step services. In a multi-step service, more than one transaction may be executed in the UTM application. This can also include distributed transaction processing (see diagram on page 111).

The calls used in the following examples are explained below:

– sign on to the UPIC carrier system \texttt{(Enable_UTM_UPIC)}
– initialize the conversation characteristics \texttt{(Initialize_Conversation)}
– establish the conversation \texttt{(Allocate)}
– send data \texttt{(Send_Data; you can also use Send_Mapped_Data)}
– receive the response \texttt{(Receive; you can also use Receive_Mapped_Data)}
– sign off from the UPIC carrier system \texttt{(Disable_UTM_UPIC)}

To simplify the diagrams in this section, the buffering of the data in the local UPIC memory during sending and receiving is not shown.
5.4.1 Communicating in a single-step UTM service

The two diagrams below show the possible forms of cooperation between a CPI-C application and a UTM application in a single-step service.

One Send and one Receive call

![Diagram](image)

**Figure 11: Single-step service with a Send/Receive call**

With a Receive call, the program waits until the response arrives from openUTM. CM_COMPLETE_DATA_RECEIVED indicates that the response has been received in full. The fact that it was the last and only message is clear from CM_DEALLOCATE_NORMAL. Instead of Send_Data and Receive, you can also use Send_Mapped_Data and Receive_Mapped_Data.
If larger volumes of data are to be transferred, several *Send* and *Receive* calls can be used when communicating in a single-step service; see the following diagram.

**Multiple Send and Receive calls**

![Diagram showing a single-step service with several Send/Receive calls](image)

- **CPI-C application**
  - `Send_Data (data1)`
  - `Send_Data (data2)`
  - `Send_Data (data3)`
  - Possibly `Prepare_To_Receive`
  - `Receive`

- **UTM application**
  - `TAC + data1`
  - Possibly `ID + password`
  - `INIT`
  - `MGET (data1)`
  - `MGET (data2)`
  - `MGET (data3)`
  - `...`
  - `MPUT NT(response1)`
  - `MPUT NE(response2)`
  - `PEND FI`

**Receive results:**
- `CM_COMPLETE_DATA_RECEIVED`
- `CM_OK`
- `CM_NO_STATUS_RECEIVED`
- `response1` in receive buffer

**Receive**

**Receive results:**
- `CM_COMPLETE_DATA_RECEIVED`
- `CM_DEALLOCATED_NORMAL`
- `response2` in receive buffer

**Disable_UTM_UPIC**

**Figure 12:** Single-step service with several Send/Receive calls

A separate *Receive* call is issued for each MPUT call.

After the first *Receive* call, CM_NO_STATUS_RECEIVED together with CM_OK indicates that further messages are pending. A second *Receive* call is thus required, which fetches the second and last message. The fact that it was the last message is clear from CM_DEALLOCATED_NORMAL.
5.4.2 Communicating in a multi-step UTM service

The diagram below illustrates one possible form of cooperation between a CPI-C application and a UTM application in a multi-step service. Data is sent and received several times in this example.

- **Enable_UTM_UPIC**
- **Initialize_Conversation**
  - possibly
  - **Set_TP_Name** or
  - **Set_Conversation_Security_Type**
  - **Set_Conversation_Security_User_ID**
  - **Set_Conversation_Security_Password**
- **Allocate**
- **Send_Data (message1)**
- **Receive**
  - **...**
  - **Receive results:**
    - CM_COMPLETE_DATA_RECEIVED
    - CM_SEND_RECEIVED
    - CM_OK
    - response1 in receive buffer
- **Processing of response1**
- **Send_Data (message2)**
- **Receive**
  - **...**
  - **Receive results:**
    - CM_COMPLETE_DATA_RECEIVED
    - CM_DEALLOCATED_NORMAL
    - response2 in receive buffer
- **Disable_UTM_UPIC**

**Figure 13: Multi-step service**

Communication in a multi-step service is required if the first response must be processed in the CPI-C application before the second message is sent to UTM.
5.4.3 Communicating in a multi-step UTM service with distributed transaction processing

The diagram below illustrates one possible form of cooperation between a CPI-C application and a UTM application in a multi-step service. In this example, distributed transaction processing (DTP) is initiated on the UTM side between two UTM applications.

<table>
<thead>
<tr>
<th>CPI-C application</th>
<th>UTM application</th>
<th>UTM application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable_UTM_UPIC</td>
<td>Initialize_conversation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>possibly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set_TP_Name or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set_Conversation_xxx...</td>
<td></td>
</tr>
<tr>
<td>Allocate</td>
<td>Send_Data (message)</td>
<td>Receive</td>
</tr>
<tr>
<td></td>
<td>Receive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send_Data (message)</td>
<td>Receive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable_UTM_UPIC</td>
<td>Initialize_conversation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>possibly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set_TP_Name or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set_Conversation_xxx...</td>
<td></td>
</tr>
<tr>
<td>Allocate</td>
<td>Send_Data (message)</td>
<td>Receive</td>
</tr>
<tr>
<td></td>
<td>Receive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Possibly ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive results:</td>
<td>- CM_COMPLETE_DATA_RECEIVED</td>
<td>- CM_DEALLOCATED_NORMAL</td>
</tr>
<tr>
<td></td>
<td>- response in receive buffer</td>
<td></td>
</tr>
<tr>
<td>Disable_UTM_UPIC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 14: Multi-step service with DTP
5.4.4 Querying the transaction state

The openUTM application sends information on the transaction and service state to the client with each user message. The CPI-C application can read this information using the `Extract_Transaction_State` call.

The state information is sent in a 4-byte field. The first two bytes indicate the state of the service and transaction, the second two bytes supply diagnostics information, see section “Extract_Transaction_State - Querying service and transaction state of the server” on page 175. The program can therefore detect, for example,

– whether the processing step was completed with or without transaction termination,
– whether the service was also terminated, or
– whether the transaction was rolled back.

The CPI-C program can respond appropriately and, for example, provide detailed information on whether input was accepted successfully or whether input must be re-sent to the server because the transaction was rolled back.
5.5 User concept, security and restart

With the UPIC carrier system, the UTM user concept can be used on the CPI-C and XATMI interface. In this case, important openUTM security functions and restart functions relevant for data security are available with client/server communication.

5.5.1 User concept

In a UTM application, it is possible to generate UTM user IDs and protect them by passwords of a particular complexity level. These user IDs and passwords with their complexity levels must be generated in the UTM application with USER statements. Each user ID generated for a UTM application can be used both by a client program and by a terminal user.

The user concept implemented on the CPI-C and XATMI interface is valid for the duration of a conversation, i.e. each time a conversation is established the program must transfer the authorization data (user ID and possibly password) to openUTM. As of openUTM V5.0, a client program can also sign on using a sign on service (SIGNON service; see the openUTM manual "Programming Applications with KDCS").

Multiple sign-ons under one UTM user ID

If a UTM user ID is generated with service restart (USER ..., RESTART=YES), openUTM links the UTM user ID with a restartable service context which is implicitly assigned using the user ID.

Only one client program or one terminal user can work with the UTM application at any one time under this type of UTM user ID.

If, in an application which allows multiple sign-ons under a user ID (SIGNON ..., MULTI-SIGNON=YES), a UTM user ID is generated without restart (USER ..., RESTART=NO), then multiple sign-ons are possible under this user ID. The restartable service context is not required in this case.
5.5.2 Security functions

The following security functions are implemented in UTM:

- System access control functions

These functions are implemented in openUTM by UTM user IDs and passwords of a particular complexity level. The functions are used as follows in CPI-C and XATMI:

- The following calls are available in CPI-C:
  
  \[\text{Set\_Conversation\_Security\_Type}\]: define type of system access control
  
  \[\text{Set\_Conversation\_Security\_User\_ID}\]: specify UTM user ID
  
  \[\text{Set\_Conversation\_Security\_Password}\]: specify associated password

- In addition with UPIC

  \[\text{Set\_Conversation\_Security\_New\_Password}\]: assign a new password

  You must issue these calls before the conversation is established.

  If sign-on was unsuccessful, the following call is also available after a \text{Receive} or \text{Receive\_Mapped\_Data} call:

  \[\text{Extract\_Secondary\_Return\_Code}\]: query the secondary return code

- On the XATMI interface, the \text{tpinit()} call has corresponding parameters with which these system access control functions are activated (see page 284).

As soon as the CPI-C or XATMI program uses these calls, the system access control functions and data security functions outlined below become effective implicitly.

- Data access control functions

In order to make certain services of the UTM server application accessible to a select group of users only, you can use the key code/lock code concept or the access list concept of openUTM (see the openUTM manual “Concepts und Functions”).

- In the lock/key code concept lock codes can be assigned to the transaction codes (services) and the LTERM partners of the UTM server application. These objects can only be accessed by users or clients whose user IDs are assigned the corresponding key codes. At generation time, a key set with one or more key codes is assigned to the user ID (USER ...,KSET=key-set-name). The key set defines which services of the UTM application can be accessed by the client.

- In the access list concept roles are defined as key codes. The transaction codes are protected using access lists. One or more roles are assigned to each user ID (generation statement USER ...,KSET=). A client may not access a service using a specific user ID unless at least one of the roles of the user ID is included in the access list. Roles can also be assigned to LTERM partners; the same then applies for access using an LTERM partner.
● Data security through user-specific long-term storage area (ULS)

A user-specific long-term storage area can be assigned to each UTM user ID at generation. This storage area can only be accessed by program units of the user/client as well as programs started by the administrator, whereby conflicting accesses are prevented by openUTM. The information in the ULS is retained even after the conversation is terminated. It is not deleted, but can only be overwritten by blank messages. The ULS is used to transfer data between conversations and the user’s programs.

A user-specific long-term storage area is assigned to each user ID of the UTM application with the KDCDEF control statement ULS.

Security functions in the client/server environment are implemented as follows within openUTM:

1. Before a UTM service is started, the authorization data coming from the client is validated and the corresponding UTM user ID is assigned, together with the associated key set. This corresponds roughly to a KDCSIGN of a terminal user immediately before the service starts.

   Sign-on is still possible if the validity period of the user password has expired but the UTM application is generated with Grace Sign-On, see page 116.

2. If the lock/key code or access list concept is used, openUTM checks whether the service may be started under this user ID and using this LTERM partner. If so, in the UTM service, the UTM user ID transferred from the client appears in the header of the communication area (KB header). The authorizations (key sets) linked with this UTM user ID apply.

3. The ULS block assigned to the UTM user ID transferred from the client can be used. If several clients sign on under one user ID, they share usage of the same ULS block, as there is only ever one ULS block for each user ID.

4. At the end of the service, the assignment (points 1 through 3) is canceled again.
Sign-on after expiry of the password validity period (Grace Sign-On)

If the UTM application is generated with Grace Sign-On, a client may still sign on to the application after expiry of the password validity period. If no sign-on service is generated for the UPIC client, the program is supplied with the return code CM_SECURITY_NOT_VALID after a Receive or Receive_Mapped_Data call. For UTM applications as of Version V5.1A30 additional information is supplied in the form of a secondary return code. If the password has expired, this code contains one of the following values:

- CM_SECURITY_PWD_EXPIRED_RETRY if the application is generated with Grace Sign-On. In this case the program can set a new password using Set_Conversation_Security_New_Password at the next sign-on. The new password must differ from the old password but must satisfy the same requirements (length, complexity, use of special characters).

- CM_SECURITY_PWD_EXPIRED_NO_RETRY if the application is not generated with Grace Sign-On. In this case the client user can no longer sign on using this UTM user ID. He or she must request the administrator of the UTM application to issue a new password.

The secondary return code of a Receive or Receive_Mapped_Data call can also be queried using a subsequent CPI-C Extract_Secondary_Returncode call. Extract_Secondary_Returncode supplies the secondary return code of the last Receive or Receive_Mapped_Data call.
5.5.3 Restart

A true restart is only possible with the CPI-C interface from UPIC, because only this interface can communicate in multi-step UTM services. However, the last output message can also be read with the XATMI interface; see section “Restart” on page 277. The following description therefore only refers to CPI-C client programs.

A service context is linked with the UTM user ID. Amongst other things, the service context contains the last output message and service data such as KB and LSSBs, etc. The client can also send a client context to the UTM application, see section “Restart with client context” on page 119.

Restart capability depends on how a UTM user ID is generated:

– If a UTM user ID is generated with USER ...,RESTART=YES (default value), openUTM performs a service restart after system failure or after loss of the connection to the client. In other words, openUTM reactivates the service context and, where appropriate, the client context for the user ID.

– If a UTM user ID is generated with RESTART=NO, openUTM does not implement any service restarts, even if the LTERM partner used by the client is generated with LTERM ...,RESTART=YES.

A service restart means that after the client signs on again, processing continues at the last synchronization point of a service which is still open. openUTM retransmits the last message of the open service and, where appropriate, the client context to the client. The client can then continue the service.

If an open service exists for the client under the user ID, this service must be continued immediately after the next sign-on, as otherwise openUTM terminates the open service abnormally.

The client program must initiate the restart by first of all establishing a new conversation and transferring the KDCDISP transaction code in the Set_TP_Name call. The example below illustrates this type of “restart program” for CPI-C.
Example

Initialize_Conversation (...
Set_Conversation_Security_Type (...,CM_SECURITY_PROGRAM,...) 1.
Set_Conversation_Security_User_ID (...),"UTMUSER1"...,) 1.
Set_Conversation_Security_Password (...),"SECRET"...,) 1.
Set_TP_Name (...),"KDCDISP"...,) 2.
Allocate (...)

Send_Data (...) 3.
      /* blank message */
Receive (...)

      return_code=CM_OK
            /* service open, send authorization transferred to client */
            /* continue communication in UTM service */
      status_received=CM_SEND_RECEIVED 4.
            or
      return_code=CM_DEALLOCATED_NORMAL 5.
            /* end of service, restart terminated */
            or
      return_code=CM_TP_NOT_AVAILABLE_NO_RETRY 6.
            /* restart not possible */

1. The program uses the system access control functions of openUTM and explicitly sets the UTM user ID and password.
2. The program must set the TP_name to KDCDISP for the restart.
3. No data can be sent with Send_Data, i.e. send_length must be set to 0 (“blank message”).
4. Processing and communication with the UTM service can be continued.
5. The program has already received the last output message; there are no more open services on the UTM side.
6. A restart is not possible, due to UTM regeneration.

The client always receives the last output message of openUTM with Receive as the result of this type of restart program.
A user can sign on to a UTM server under a particular user ID in one of several ways:

- from a terminal
- via a transport system client
- via a client program with various carrier systems

A restart by a client program is only possible if the user ID was also last used by a client program with the same carrier system. If this is not the case, openUTM rejects the client programs’ attempt to sign on (CM_SECURITY_NOT_VALID) because the open service must first be terminated by the partner that started it.

If no open service exists when the conversation is established with KDCDISP, openUTM terminates the conversation after sending the last output message of the previous service. If the last service was started by a different partner, openUTM does not transfer any messages (return code CM_TP_NOT_AVAILABLE_NO_RETRY).

To avoid these problems, a UTM user ID generated with RESTART=YES should be used either only by client programs with the same carrier system, or only by terminal users.

If no application context exists following a regeneration of the UTM application, the program receives the return code CM_TP_NOT_AVAILABLE_NO_RETRY. openUTM terminates the conversation.

Open services of a client with restart capability are transferred by the UTM utility KDCUPD as of UTM version 5.1.

### Restart with client context

With each user message the client can send what is known as a client context to the UTM application. A client context consists of a string up to 8 bytes long. The string may contain, for example, the time or a message ID.

If the user ID is generated with RESTART=YES, the client context is buffered by openUTM until the end of the conversation unless it is overwritten with a new context.

If the client requests a restart, openUTM transfers the client context to the client together with the last dialog message. By referring to the client context the program is able to uniquely identify at which point in the dialog a restart must be made and how the program must respond; for example, by outputting a specific form. The following UPIC calls are available to set and read the client context:

- **Set_Client_Context**: set client context
- **Extract_Client_Context**: output the last client context sent by openUTM
5.6 Encryption

Clients often access UTM services via open networks. There is, therefore, the possibility that unauthorized persons on the line can monitor and, for example, discover passwords for UTM user IDs or sensitive user data. In order to avoid this, openUTM supports the encryption of passwords and user data for client connections.

Encryption in openUTM can be used to control access from clients and also access to certain services. openUTM uses a combination of a symmetric AES or DES key and an asymmetric RSA key.

Encryption methods

Passwords and user data on a connection are encrypted with a symmetric key. This is either an AES or a DES key. Client and UTM application use the same symmetric key to encrypt and decrypt messages. This key is generated by the client and is transferred to the UTM application when a connection is established. The key is used for this connection only.

To increase the level of security, the AES or DES key itself is transferred in encrypted form. For this purpose, one or more RAS key pairs are created for the UTM application at generation. An RSA key pair consists of a public and a private key. The public key is transferred to the client as soon as the connection is set up by the UTM application. The client uses it to encrypt the AES or DES key. To decrypt this key, the UTM application uses the private key which is known only to the UTM application.

Up to and including openUTM Version 5.1 an RSA key of modulo length 200 was created. As of openUTM Version 5.2 up to four different RSA key pairs are created, depending on generation:

- RSA keys of modulo length 512, 1024 and 2048
- RSA keys of modulo length 200 (for reasons of compatibility with UPIC clients < V5.2).

With the help of these keys different encryption levels can be defined in UTM generation (ENCEYPTION-LEVEL operand), see table.

<table>
<thead>
<tr>
<th>Generated encryption level</th>
<th>Modulo length of the RSA key</th>
<th>Symmetric key</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUSTED</td>
<td>No key</td>
<td>No key</td>
</tr>
<tr>
<td>NONE</td>
<td>Depending on situation</td>
<td>Depending on situation</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>DES</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>AES</td>
</tr>
<tr>
<td>3</td>
<td>1024</td>
<td>AES</td>
</tr>
<tr>
<td>4</td>
<td>2048</td>
<td>AES</td>
</tr>
</tbody>
</table>

Table 7: Generated encryption levels and associated keys
In openUTM each RSA key pair can be modified and activated using administration facilities. Only activated RSA keys are used. The UPIC client can also store the public key locally in advance. When a connection is set up, the public key received is checked against the stored public key.

The active RSA key can be read out using calls of the UTM administration interface or using the openUTM WinAdmin administration tool (V5.0 or higher). The key can also be deleted using openUTM WinAdmin (V5.2 or higher).

Requirements

A requirement for encryption between openUTM and UPIC clients is that an encryption license is available at both sides. For legal reasons the encryption functions of openUTM are shipped as a separate product (openUTM-Crypt) that must be installed separately.

If an encryption level of 1 to 4 is generated for the partner in openUTM but the encryption requirements have not been satisfied, no connection is set up. This may be for one of the following reasons.

- The client does not support encryption because the encryption functionality is not installed or because the UPIC version is an export version.
- openUTM itself cannot perform encryption because it does not have a suitable encryption library (export version).

Procedure

When the client attempts to connect to the UTM application, it informs openUTM whether it supports encryption.

Once the connection between the client and the server has been established and if encryption is supported by both partners, the client sends information to the server indicating the level up to which it supports encryption. The server compares this with the generation information for the partner.

Depending on the encryption level the client generates in the UTM application, various situations can occur.

**ENCRYPTION-LEVEL=TRUSTED**

The client is generated as trusted. In this case openUTM does not request encryption. Neither can the client force encryption.
ENCRIPTION-LEVEL=None

In this case the UTM application sends the RSA key with maximum modulo length to the client. The RSA key determines the encryption level.

Depending on the length of the RSA key received the client generates an AES key (if the RSA key length $\geq 512$) or a DES key (if the RSA key length = 200). The client encrypts the AES or DES key with the RSA key and returns it to the server. openUTM stores the key for later use on this connection.

By default only passwords are encrypted.

However, the client can force encryption of user data by using the ENCRYPTION_LEVEL keyword in the upicfile or by means of the Set_Conversation_Encryption_Level call.

Notes

- If encryption functionality is not installed, passwords and user data are exchanged without encryption.

- If the encryption functionality of V5.1 is still being used because, for example, an older UPIC version (V 5.0 or 5.1) was not correctly replaced with Version 5.2, encryption level 1 is always used (RSA key of modulo length 200, DES encryption).

ENCRIPTION-LEVEL=1

The public RSA key of modulo length 200 is sent to the client. The client generates a DES key, encrypts it with the RSA key and sends it back. openUTM stores the DES key for later use.

Passwords and user data are encrypted.

The Set_Conversation_Encryption_Level call or an ENCRYPTION_LEVEL entry in the upicfile has no effect.
ENCRYPTION-LEVEL=2, 3 or 4
The UTM server sends the public RSA key associated with the appropriate encryption level. The length of this key is 512, 1024 or 2048, see table 7 on page 120.

The client generates an AES key, encrypts it with the RSA key and sends it back to the server. openUTM stores the AES key for later use on this connection.

Passwords and user data are encrypted.

The Set_Conversation_Encryption_Level call or an ENCRYPTION_LEVEL entry in the upicfile has no effect.

The client-level encryption level of the conversation can be read out using the Extract_Conversation_Encryption_Level call, preferably after the Allocate call.

Encryption with protected TAC

A service of a UTM application can be protected by assigning an encryption level to the associated TAC in the ENCRYPTION-LEVEL=tac-level operand at generation. This ensures that a client cannot call the protected service unless data is transferred with the specified encryption. The following situations can occur depending on the generation of the client and on the encryption level of the TAC.

TRUSTED is generated for the client
- openUTM does not request encryption and the client can also start protected services. The client cannot force encryption because no keys were exchanged.

NONE is generated for the client
- openUTM does not request encryption.

If a client-level encryption level > 0 was established at connection setup and if a conversation whose TAC requires level 1 or level 2 encryption is initialized, there are the following possibilities.

- client-level ≥ tac-level
  The client for this conversation has activated encryption.
  
  The service can be started. The client sends user data in an encrypted form right from the beginning.

- client-level ≥ tac-level
  The client for this conversation has not activated encryption and has not yet sent any user data.
  
  The service can be started. The UTM application transmits all output on the client-level encryption level to the client in an encrypted form. The client also encrypts all subsequent messages to openUTM on the client-level encryption level.
– *client-level < tac-level*
  The UPIC client has already sent user data that was either not encrypted or was encrypted with a lower encryption level.

  openUTM ends the conversation.

1, 2, 3 or 4 is generated for the client

If a conversation whose TAC requires level 1 or level 2 encryption (*tac-level*) is initialized, there are the following possibilities.

– *client-level ≥ tac-level*
  The service can be started.

– *client-level < tac-level*
  The service cannot be started and openUTM ends the conversation.

Note that for the connection between client and server (and therefore for all subsequent conversations on this connection) more encryption levels can be specified than for the TAC.
5.7 Multiple conversations

The multiple conversations functionality enables a CPI-C client to hold several conversations at once within a program run. The conversations can be established with different UTM server applications or the same UTM server application.

The UPIC carrier system supports multiple conversations only on systems which support multithreading (e.g. Windows and Unix systems). For more information, see page 32.

"Multiple conversations" is dependent on operation and the system. See the readme file for more information.

Multithreading means that several threads can be started within the process in which a program is running. Threads are program segments running in parallel within a process, in which processing steps are processed independently of each other. Threads are therefore often called concurrent processes. The use of threads is equivalent to a type of multiprocessing that is administered by the program itself and is executed in the same process as the program itself.

CPI-C clients which run on systems with multithreading and are implemented accordingly can therefore be connected to several UTM services at the same time.

CPI-C clients which run on systems that do not support multithreading can only hold one conversation at a time. Only when this conversation is shut down can a new one be established.

If a client application wants to process several conversations at once, each one of these conversations must be processed in a separate thread independently of the others. Here you must note the following:

- The first thread of the process in which the other threads are started is the main thread. A conversation can also be established in the main thread, as in any other process.
- For each additional conversation that the program is to establish and process in parallel, a thread must be started explicitly. System calls are provided for starting the threads. These system calls are dependent on the operating system and on the compiler used (see example on page 128).
- In each of the started threads, the runtime environment for the CPI-C client must be started. For this purpose, an Enable_UTM_UPIC call must be issued in every thread. The CPI-C program can sign on in all threads with the same or with different names.
- In each individual thread the conversation characteristics must be set with an Initialize_Conversation call. The conversation is assigned a separate conversation ID by UPIC.
– Each conversation ID can only be used within the thread in which the associated conversation was initialized and established. If the conversation ID is specified in another thread in a CPI-C call, UPIC brings back the return code CM_PROGRAM_PARAMETER_CHECK.

– In each thread the program must sign off from UPIC with Disable_UTM_UPIC before the thread is terminated.

– The main thread must not terminate until all other threads have signed off and terminated.

The sequences within the client program are shown in the following diagram.

Upic local:
Upic-L does not support the “Multiple conversations” function.
Main thread

Starting of process and of main thread:
main()
...
Starting of two further threads in the process:
CreateThread()
CreateThread()

The 3 threads run parallel to and independently of each other.

Sign-on of main thread:
Enable_UTM_UPIC()
...
Initialize a conversation in main thread:
Initialize_Conversation()

Conversation ID CID0 is brought back; it is only to be used in main thread.
...
Set...(CID0,....)
Allocate (CID0,...)
Send_Data(CID0,...)
...
possibly:
Prepare_To_Receive(CID0..)
...
Receive(CID0,...)
...
Sign-off of main thread:
Disable_UTM_UPIC()

Wait for other threads to terminate:
WaitForMultiple_Objects()

Terminate main thread and process:
ExitProcess(0)

Thread1

Sign-on of thread1:
Enable_UTM_UPIC()
...
Initialize a conversation in thread1:
Initialize_Conversation()

Conversation ID CID0 is brought back; it is only to be used in thread1.
...
Set...(CID1,....)
Allocate (CID1,...)
Send_Data(CID1,...)
...
possibly:
Prepare_To_Receive(CID1..)
...
Receive(CID1,...)
...
Sign-off of thread1:
Disable_UTM_UPIC()

Terminate thread1:
ExitThread(0)

Thread2

Sign-on of thread2:
Enable_UTM_UPIC()
...
Initialize a conversation in thread2:
Initialize_Conversation()

Conversation ID CID0 is brought back; it is only to be used in thread2.
...
Set...(CID2,....)
Allocate (CID2,...)
Send_Data(CID2,...)
...
possibly:
Prepare_To_Receive(CID2..)
...
Receive(CID2,...)
...
Sign-off of thread2:
Disable_UTM_UPIC()

Terminate thread2:
ExitThread(0)

Figure 15: Starting several threads within a process
(the gray-hatched area corresponds to the process in which the client program is running)
Example of multiple conversations (in Visual C++ for Windows systems)

The schema belonging to the client program is structured as follows:

```c++
void main () {
  ... 
  thrd[0] = CreateThread(...,UpicThread,...);  
  thrd[1] = CreateThread(...,UpicThread,...); 
  ... 
  Enable_UTM_UPIC (...); 
  ... 
  /* Calls for establishing and processing a conversation */ 
  /* in the main thread: */ 
  Initialize_Conversation (...)
  ... 
  Allocate (...)
  .... 
  Send_Data (...)
  ...
  Receive (...)
  ...
  Disable_UTM_UPIC (...);
  ...
  WaitForMultipleObjects(2,&thrd[0],...);
  ExitProcess (0); 
}

DWORD WINAPI UpicThread(LPVOID arg) {
  ... 
  Enable_UTM_UPIC (...);
  ... 
  /* Calls for establishing and processing conversation in thread */
  /* as in main thread under 3. */
  ... 
  Disable_UTM_UPIC (...);
  ...
  ExitThread(0);
}
```
1. Process and main thread are started.

2. Two further threads are started via the corresponding system call. The system call depends on the system and compiler used.

   Each thread is started with the `UpicThread()` function. In `UpicThread()` a conversation is established and processed (see point 6 below). `UpicThread` is a freely selectable name.

3. Each thread must explicitly execute an `Enable_UTM_UPIC` call and a `Disable_UTM_UPIC` call. At this point the main thread signs on to UPIC. After the `Enable_UTM_UPIC` call the CPI-C calls can then be issued for establishing a conversation in the main thread and processing this conversation. Several conversations can be processed concurrently in the main thread. Once the conversation in the main thread has terminated, this thread must sign off with `Disable_UTM_UPIC`.

4. The main thread waits until both the threads it has started have terminated.

5. End of the process and the main thread.

6. `UpicThread()` is the function that is called when a new thread is started. In this function, the relevant thread signs on to UPIC with `Enable_UTM_UPIC` and processes "its conversation" (with `Initialize_Conversation`, `Set_...`, `Send_Data`, `Receive ...`). Here too, several conversations can be processed concurrently. When the last conversation has terminated, the thread signs off with `Disable_UTM_UPIC`.

   `UpicThread()` must be programmed such that the threads running concurrently do not interfere with each other. The code must therefore be structured so that it can be executed by several threads at the same time, i.e. the functions used must not mutually destroy the context.

7. Termination of the thread.

openUTM-Client comes with the source code for a sample program on multiple conversations (see section “Sample programs for Windows systems” on page 371).
5.8 Default server and DEFAULT name of a client

In practice it is often the case that a client communicates mainly with one particular UTM server. To simplify the configuration of UPIC clients and the programming of CPI-C client programs in such cases, you can define a DEFAULT server for your client application in the upicfile (see page 331). In order to be connected to the DEFAULT server, the client program can omit specification of a symbolic destination name when initializing the conversation with Initialize_Conversation. It transfers an empty name to UPIC and is then automatically connected to the DEFAULT server.

You can also define a service on the DEFAULT server as the DEFAULT service. To do this, you specify the transaction code of this service in the DEFAULT server entry in the upicfile. If the CPI-C program then does not specify a transaction code when initializing a conversation for the DEFAULT server (it does not call Set_TP_Name), the conversation is automatically established with the DEFAULT service. If another service is to be started on the DEFAULT server, the client program must transfer the transaction code of this service to UPIC with Set_TP_Name (e.g. TP_name=KDCDISP must be selected at service restart).

In the same way, you can define a DEFAULT name for the local CPI-C client application in the upicfile. If the client program specifies an empty local application name when the application signs on to UPIC (with Enable_UTM_UPIC), the client is signed onto UPIC with the DEFAULT name and UPIC uses the address information assigned to the DEFAULT name to establish the conversation.

If a DEFAULT name is used for the CPI-C application, it may occur that several program runs of a UPIC client want to sign on to a UTM application with the same name at the same time. This is the case if the client program is started several times in parallel or if a program wants to establish several conversations with a UTM application in parallel (multiple conversations). To enable the server application to accept these sign-ons, the conditions described in the following section must be met.
5.8.1 Multiple sign-on to the same UTM application with the same name

Multiple simultaneous sign-on by a client application to a UTM application using the same name in each case is possible.

To enable a client to sign on more than once with the same name, an LTERM pool which supports multiple sign-on under the same name must have been generated in the UTM server application for the system on which the client is running. Such an LTERM pool is generated in openUTM as follows:

TPOOL ...,CONNECT-MODE=MULTI

For the name the client uses to sign on to the UTM application (PTERM name), a PTERM statement must not be generated in the UTM application (see openUTM manual "Generating Applications"), otherwise multiple sign-on via the LTERM pool is not possible.

The CPI-C program can sign on to the UTM application via the LTERM pool as many times as there are LTERM partners available in the LTERM pool (the number is set by UTM administration). It can use the same name or different names to sign on.
5.9 CPI-C calls in UPIC

Input and output parameters and possible return codes are described below for each function.

In general, all parameters are passed at the interface by means of addresses. The symbols → and ← designate input and output parameters respectively.

The symbolic destination name and the conversation_ID are always exactly eight characters long.

The return codes supplied at the interface are independent of the transport system used. A distinction between local and remote connections is made only in the explanation of certain return codes and in notes on error messages.

Overview

The interface functions can be used on all platforms in the programming languages C, C++ and COBOL, and are provided in libraries.

The following description of the CPI-C calls has therefore been kept as language-independent as possible, even though it uses the notation of the C interface. In section “COBOL interface” on page 273 you will find a description of the special features of the COBOL interface which you must take into account when creating CPI-C programs in COBOL.

The precise function declaration is given separately for each call.

Program calls

A client communicates with a UTM server application by calling functions. These calls are used to establish the conversation characteristics and to exchange data and control information. The CPI-C calls supported by UPIC can be categorized into two groups:

- Starter-set calls
  Starter-set calls enable simple communication with a UTM server. They are used for simple data exchange processes, e.g. for accepting the initialized values of conversation characteristics.

- Advanced-function calls
  Advanced-function calls allow more specialized functions to be executed. For example, the conversation characteristics can be modified using Set calls.
Starter-set functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize_Conversation</td>
<td>Initializes conversation characteristic</td>
</tr>
<tr>
<td>Allocate</td>
<td>Starts a conversation</td>
</tr>
<tr>
<td>Deallocate</td>
<td>Ends a conversation abnormally</td>
</tr>
<tr>
<td>Send_Data</td>
<td>Sends data</td>
</tr>
<tr>
<td>Receive</td>
<td>Receives data</td>
</tr>
</tbody>
</table>

Table 8: Starter-set functions

It is assumed that the CPI-C program (client) is always the active part. For this reason the CPI-C function `Accept_Conversation` is not supported.

On systems which support multithreading (e.g. Windows, Solaris 5.7), several conversations with different UTM servers can be active at the same time in a CPI-C program. Each conversation, including the associated `Enable_UTM_UPIC` and `Disable_UTM_UPIC` calls, must be executed in a separate thread.

On all other systems, only one conversation at a time can be active in a CPI-C program.
## Advanced-function calls

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert_Incoming</td>
<td>Converts received data to the local code</td>
</tr>
<tr>
<td>Convert_Outgoing</td>
<td>Converts the data to be sent from the local code to the code of the communication partner</td>
</tr>
<tr>
<td>Deferred_Deallocate</td>
<td>Terminates the conversation as soon as the current transaction has been terminated successfully</td>
</tr>
<tr>
<td>Extract_Conversation_State</td>
<td>Inquires about the conversation state</td>
</tr>
<tr>
<td>Extract_SecondaryInformation</td>
<td>Inquires about further information</td>
</tr>
<tr>
<td>Extract_Partner_LU_Name</td>
<td>Inquires about the value of the conversation characteristics partner_LU_name</td>
</tr>
<tr>
<td>Prepare_To_Receive</td>
<td>Sends the data buffered in the send buffer to the communication partner immediately and switches to the “Receive” state</td>
</tr>
<tr>
<td>Receive_Mapped_Data</td>
<td>Receives the data together with the structure information (format identifier)</td>
</tr>
<tr>
<td>Send_Mapped_Data</td>
<td>Sends the data together with the structure information (format identifier)</td>
</tr>
<tr>
<td>Set_Conversation_Security_Password</td>
<td>Sets the password for a UTM user ID</td>
</tr>
<tr>
<td>Set_Conversation_Security_Type</td>
<td>Activates or deactivates the security function</td>
</tr>
<tr>
<td>Set_Conversation_Security_User_ID</td>
<td>Sets the UTM user ID</td>
</tr>
<tr>
<td>Set_Partner_LU_name</td>
<td>Sets the value for the conversation characteristics partner_LU_name</td>
</tr>
<tr>
<td>Set_Deallocate_Type</td>
<td>Sets values for the conversation characteristic deallocate_type</td>
</tr>
<tr>
<td>Set_Receive_Type</td>
<td>Sets values for the conversation characteristic receive_type</td>
</tr>
<tr>
<td>Set_Sync_Level</td>
<td>Sets values for the conversation characteristic sync_level</td>
</tr>
<tr>
<td>Set_TP_Name</td>
<td>Sets the name for a partner program (transaction code)</td>
</tr>
</tbody>
</table>

Table 9: Advanced Functions

1 Not a component of X/Open CPI-C version 2
### Additional UPIC functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable_UTM_UPIC</td>
<td>Signs on to the UPIC carrier system</td>
</tr>
<tr>
<td>Extract_Client_Context</td>
<td>Outputs the client context</td>
</tr>
<tr>
<td>Extract_Communication_Protocol</td>
<td>Queries the UPIC protocol version</td>
</tr>
<tr>
<td>Extract_Conversation_Encryption_Level</td>
<td>Inquires about encryption level</td>
</tr>
<tr>
<td>Extract_Conversion</td>
<td>Queries the ASCII-EBCDIC conversion</td>
</tr>
<tr>
<td>Extract_Cursor_Offset</td>
<td>Inquires about cursor position offset</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>Queries secondary return codes</td>
</tr>
<tr>
<td>Extract_Transaction_State</td>
<td>Queries the service and transaction state of the server</td>
</tr>
<tr>
<td>Disable_UTM_UPIC</td>
<td>Signs off from the UPIC carrier system</td>
</tr>
<tr>
<td>Set_Allocate_Timer</td>
<td>Setting timer for the Allocate call</td>
</tr>
<tr>
<td>Set_Client_Context</td>
<td>Sets the client context</td>
</tr>
<tr>
<td>Set_Communication_Protocol</td>
<td>Sets the UPIC protocol version</td>
</tr>
<tr>
<td>Set_Conversion</td>
<td>Sets the ASCII-EBCDIC conversion</td>
</tr>
<tr>
<td>Set_Conversation_Encryption_Level</td>
<td>Sets encryption level</td>
</tr>
<tr>
<td>Set_Conversation_Security_New_Password</td>
<td>Sets a new password for a UTM user ID</td>
</tr>
<tr>
<td>Set_Function_Key</td>
<td>Sets the value of the function key to be transferred</td>
</tr>
<tr>
<td>Set_Receive_Timer</td>
<td>Sets the timeout timer for the blocking receive of data</td>
</tr>
<tr>
<td>Set_Partner_Host_Name</td>
<td>Sets the host name of the partner application</td>
</tr>
<tr>
<td>Set_Partner_IP_Address</td>
<td>Sets the IP address of the partner application</td>
</tr>
<tr>
<td>Set_Partner_Port</td>
<td>Sets the TCP/IP port of the partner application</td>
</tr>
<tr>
<td>Set_Partner_Tsel</td>
<td>Sets the TSEL of the partner application</td>
</tr>
<tr>
<td>Set_Partner_Tsel_Format</td>
<td>Sets the TSEL format of the partner application</td>
</tr>
<tr>
<td>Specify_Local_Tsel</td>
<td>Sets the TSEL of the local application</td>
</tr>
<tr>
<td>Specify_Local_Tsel_Format</td>
<td>Sets the TSEL format of the local application</td>
</tr>
<tr>
<td>Specify_Local_Port</td>
<td>Sets the TCP/IP port of the local application</td>
</tr>
<tr>
<td>Specify_Secondary_Return_Code</td>
<td>Sets the properties of the secondary return code</td>
</tr>
</tbody>
</table>

Table 10: Additional UPIC Functions
Allocate - Establishing a conversation

A program uses the Allocate (CMALLC) call to establish a conversation with a UTM application. The name of the CPI-C program is specified in the preceding Enable_UTM_UPIC call.

Syntax

CMALLC (conversation_ID, return_code)

Parameters

→ conversation_ID  Identifier of the initialized conversation (supplied by the Initialize call).

← return_code  Result of the function call.

Result (return_code)

CM_OK  The call is OK.

CM_ALLOCATE_FAILURE_RETRY  UPIC-L  The conversation cannot be established due to a temporary resource bottleneck. Check the error message for the local UTM application as well.

CM_ALLOCATE_FAILURE_NO_RETRY  Possible causes:
  – The conversation cannot be established due to an error, e.g. the transport connection to the UTM application could not be set up.
  – The transport connection was rejected at the UTM end because in the UTM application a TPOOL or PTERM connecting point is defined with ENCRYPTION_LEVEL=1 (or 2, 3, 4), but the add-on product openUTM-CRYPT is not installed.
  – The transport connection was rejected at the UTM end because in the UTM application a TPOOL or PTERM connecting point is defined with ENCRYPTION_LEVEL=NONE and the called TAC with ENCRYPTION_LEVEL=1 (or 2), but the add-on product openUTM-CRYPT is not installed.

CM_OPERATION_INCOMPLETE  The call was interrupted by the expiry of the timer set using Set_Allocate_Timer.
CM_PARAMETER_ERROR
A TAC was not specified in the upicfile or in a Set_TP_Name call, or the conversation_security_type is CM_SECURITY_PROGRAM and the security_user_ID characteristic is not set.

CM_PROGRAM_STATE_CHECK
The call is not permitted in the current state.

CM_PROGRAM_PARAMETER_CHECK
The value for conversation_ID is invalid.

CM_PRODUCT_SPECIFIC_ERROR
- There is a protocol error.
- During operation without TNS the host name (as specified using Set_Partner_Host_Name or in the upicfile) is not defined in the hosts file.
- For this conversation, there is an RSA key stored in the upicfile; this key differs in either content or length from the received RSA key.

CM_SECURITY_NOT_SUPPORTED
- The partner application does not support the desired security_type.
- A new password has been set, but the partner application with which a conversation has been established is a UTM version prior to version V5.1 and does not support password changes for the UPIC-Client.

State change
- If the return code is CM_OK, the conversation is established and the program enters the “Send” state.
- If the return code is CM_ALLOCATE_FAILURE_RETRY/NO_RETRY or CM_SECURITY_NOT_SUPPORTED, the program enters the “Reset” state.
- In all other error situations, the program does not change its state.

Notes
- If the UTM application rejects initiation of the service, e.g. due to an invalid transaction code, this is not reported until the next Receive call is issued.
- If the specified user ID was not generated in the UTM application, or if an incorrect password or no password was sent for a generated user ID, this is not reported until the next Receive call is issued.
Behavior in the event of errors

CM_ALLOCATE_FAILURE_RETRY
Temporary resource bottleneck has occurred during the conversation.
*Initialize_Conversation*, followed by the *Allocate* call.

CM_ALLOCATE_FAILURE_NO_RETRY
Reboot the UTM application or generate the PTERM specified in
*Enable_UTM_UPIC* for openUTM. You may need to install the encryption module as
well or change the encryption level.

CM_PARAMETER_ERROR
Add a TAC to the entry for the current *sym_dest_name* or specify a TAC with the
*Set_TP_Name* call.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
  - Store either a valid RSA key or no key at all.
  - Notify the service department and produce diagnostic report

Function declaration: Allocate

```
CM_ENTRY Allocate ( unsigned char CM_PTR conversation_ID,
                     CM_RETURN_CODE CM_PTR return_code)
```
Convert_Incoming - Converting data from code of sender to local code

With the UPIC carrier system on Unix systems and Windows systems, the Convert_Incoming (CMCNVI) call converts EBCDIC data to the code used locally on the machine.

With the UPIC carrier system under BS2000/OSD, Convert_Incoming converts the data from ASCII to the code used locally on the BS2000 computer.

Syntax

CMCNVI (data, length, return_code)

Parameters

↔ data Address of the data to be converted. The data is then overwritten by the converted data.

→ length Length of the data to be converted.

← return_code Result of the function call.

Result (return_code)

CM_OK The call is OK.

State change

This call does not change the program state.

Notes

● The data must be in printable form.

● The relevant conversion table is stored on

  – on Unix systems and Windows systems in the file upic-dir\kcsaeea.c.
  – on BS2000 systems in the library $userid.SYSLIB.UTM-CLIENT.060

Function declaration: Convert_Incoming

    CM_ENTRY Convert_Incoming ( unsigned char CM_PTR string,
    CM_INT32 CM_PTR string_length,
    CM_RETURN_CODE CM_PTR return_code)
Convert_Outgoing - Converting data from local code to code of receiver

With the UPIC carrier system on Unix systems and Windows systems, the Convert_Outgoing (CMCNVO) call converts data from the code used locally on the machine to EBCDIC.

With the UPIC carrier system under BS2000, Convert_Outgoing always converts the data from the code used locally to ASCII.

Syntax

CMCNVO (data, length, return_code)

Parameters

↔ data Address of the data to be converted. The data is then overwritten by the converted data.

→ length Length of the data which are converted.

← return_code Result of the function call.

Result (return_code)

CM_OK

The call is OK.

State change

This call does not change the program state.

Notes

● The data must be in printable form.

● The relevant conversion table is stored on

  – Unix systems and Windows systems in the file upic-dir\kcsaea.c

  – on BS2000 systems in the library $userid.SYSLIB.UTM-CLIENT.060

Function declaration: Convert_Outgoing

CM_ENTRY Convert_Outgoing ( unsigned char CM_PTR string,  
CM_INT32 CM_PTR string_length,  
CM_RETURN_CODE CM_PTR return_code)
Deallocate - Terminating a conversation

A CPI-C program uses the *Deallocate* (CMDEAL) call to end a conversation abnormally. After the call has been executed successfully, the `conversation_ID` is no longer assigned to a conversation. Normally, a conversation is always ended together with the UTM process. Termination of a conversation by the CPI-C program is always regarded as abnormal. The value of `deallocate_type` must therefore be set to CM_DEALLOCATE_ABEND by the `Set_Deallocate_Type` (CMSDT) call before a `Deallocate` call is issued.

Syntax

CMDEAL (conversation_ID, return_code)

Parameters

→ conversation_ID  Identifier of the conversation to be ended.
← return_code  Result of the function call.

Result (*return_code*)

CM_OK  
The call is OK.
CM_PROGRAM_STATE_CHECK  
The call is not permitted in the current state.
CM_PROGRAM_PARAMETER_CHECK  
The value of `conversation_ID` is invalid.
CM_PRODUCT_SPECIFIC_ERROR  
The value of `deallocate_type` has not been set to CM_DEALLOCATE_ABEND by a preceding `Set_Deallocate_Type` call.

State change

If the return code is CM_OK, the program enters the “Reset” state. In all other error situations, the program does not change its state.
Behavior in the event of errors

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
Modify the program and incorporate the Set_Deallocate_Type call.

Function declaration: Deallocate

CM_ENTRY Deallocate ( unsigned char CM_PTR conversation_ID,
                  CM_RETURN_CODE CM_PTR return_code)
Deferred_Deallocate - Terminating a conversation after termination of a transaction

A CPI-C program uses the *Deferred_Deallocate* (CMDFDE) call to terminate the conversation as soon as the current transaction is successfully terminated. The call can be used at any time within a transaction. *Deferred_Deallocate* serves only to make CPI-C programs more portable. It does not change the state of the program.

**Syntax**

CMDFDE (conversation_ID, return_code)

**Parameters**

→ conversation_ID  Identifier of the conversation to be terminated.
← return_code  Result of the function call.

**Result (return_code)**

CM_OK
The call is OK.

CM_PROGRAM_PARAMETER_CHECK
The value of conversation_ID is invalid.

CM_PROGRAM_STATE_CHECK
The program is in “Start” state.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

**State change**

This call does not change the program state.
Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide enough memory for the internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

CM_PROGRAM_STATE_CHECK
Modify program

Function declaration: Deferred_Deallocate

CM_ENTRY Deferred_Deallocate ( unsigned char CM_PTR conversation_ID,
    CM_RETURN_CODE CM_PTR return_code)
**Disable_UTM_UPIC - Signing off from the UPIC carrier system**

A program uses the `Disable_UTM_UPIC` (CMDISA) call to sign off from the UPIC carrier system. After the call has been successfully executed, no further CPI-C calls are permitted. If another connection exists for the program, it is cleared down. In addition, the program signs off from the transport system.

This call must be the last call of a CPI-C program. It is not needed if you continue with a further `Initialize` call after ending the conversation.

This function is not included in the CPI-C interface, but is one of the additional UPIC functions.

**Syntax**

CMDISA (local_name, local_name_length, return_code)

**Parameters**

→ local_name  Name of the program, i.e. the name specified in the preceding `Enable_UTM_UPIC` call.

→ local_name_length  Length of `local_name`.

Minimum: 0, maximum: 8

`local_name_length`=0 means that an “empty local application name” is transferred (see section “Enable_UTM_UPIC - Signing on to the UPIC carrier system” on page 147)

← return_code  Result of the function call.

**Result (return_code)**

CM_OK  
The call is OK.

CM_PROGRAM_STATE_CHECK  
The call is not permitted in the current state.

CM_PROGRAM_PARAMETER_CHECK  
The program is not signed on to UPIC with `local_name`, or the value of `local_name_length` is < 1 or > 8.

CM_PRODUCT_SPECIFIC_ERROR  
An error occurred when signing off from UPIC or when clearing down the connection.
State change

If the return code is CM_OK, the program is signed off and enters the “Start” state. In all other error conditions, the program does not change its state.

Note

You must use this call if you wish to terminate the process with `exit()` in the event of an error condition in the application program.

For performance reasons, this function should only be called immediately before the process is terminated, provided no error has occurred.

Behavior in the event of errors

CM_PRODUCT_SPECIFIC_ERROR
   Notify the service department and produce diagnostic report.

CM_PROGRAM_STATE_CHECK
   Modify program.

CM_PROGRAM_PARAMETER_CHECK
   Modify program.

Function declaration: Disable_UTM_UPIC

CM_ENTRY Disable_UTM_UPIC ( unsigned char CM_PTR local_name,
   CM_INT32 CM_PTR local_name_length,
   CM_RETURN_CODE CM_PTR return_code)
Enable_UTM_UPIC - Signing on to the UPIC carrier system

This call must be issued before other CPI-C calls are used. The *Enable_UTM_UPIC* (CMENAB) call enables a program to sign on to the UPIC carrier system using its own name. The name serves to establish the connection between the UTM service and the CPI-C program (see also section “Initialize_Conversation - Initializing the conversation characteristics” on page 178).

In the *upicfile*, you can define a default name for the CPI-C application (LN.DEFAULT entry; see page 341). If the CPI-C program is to sign on to the UPIC carrier system with this default name, it can transfer an “empty local application name” in the *local_name* field. UPIC then searches in the *upicfile* for the LN.DEFAULT entry and uses the corresponding application name to establish the connection to the UTM service. Several CPI-C program runs can sign on with the default name simultaneously and also establish conversations to the same UTM service.

After the *Enable_UTM_UPIC* call has been executed successfully, the program is provided with an intact runtime environment. After this call is issued, changes in the *upicfile* do not come into effect for the program until the next *Enable_UTM_UPIC* call.

This function is not included in the CPI-C interface, but is one of the additional UPIC functions.

Syntax

```
CMENAB (local_name, local_name_length, return_code)
```

Parameters

- **local_name** Name of the program.

  The following specifications are possible (see also section “Side information for the local application” on page 339):

  - **with UPIC remote**:
    - Local application name defined in the *upicfile*.
    - Name under which the program is entered in the TNS directory or is known in CMX.
    - Any name, the TNS properties of which can still be modified using the following *Specify* calls.
    - Empty local application name.

    The program then signs on to UPIC under the DEFAULT name of the CPI-C application, provided that an LN.DEFAULT name exists in the *upicfile* at the time of the call.
with UPIC local:
- PTERM name by which the client is known in the configuration of the UTM application.
- Local application name defined in the upicfile.
- If an LTERM pool for the partner type UPIC-L (TPOOL with PTYPE=UPIC-L) exists in the UTM partner application, you can specify any name of up to 8 characters for local_name.
- Empty local application name.

The prerequisite is that an LN.DEFAULT entry exists in the upicfile at the time of the call.

You can transfer an empty local application name by:
- transferring 8 blanks in local_name and setting local_name_length=8
- setting local_name_length=0.

If you transfer an empty application local name, UPIC takes the application name of the LN.DEFAULT entry to establish the connection to the UTM partner application.

→ local_name_length Length of local_name
Minimum: 0, maximum: 8

If a local application name from the upicfile is entered in local_name, then local_name_length=8 must be specified.

If you specify local_name_length=0, the contents of the local_name field will be ignored, i.e. local_name will be treated as an “empty local application name”. An LN.DEFAULT entry must exist in the upicfile.

← return_code Result of the function call
Result (return_code)

CM_OK
   The call is OK.

CM_PROGRAM_STATE_CHECK
   The program is already signed on to UPIC.

CM_PROGRAM_PARAMETER_CHECK
   Possible causes:
   – the value of local_name_length is less than 1 or greater than 8
   – there is not enough internal memory available, or
   – an attempt to access the upicfile has failed

CM_PRODUCT_SPECIFIC_ERROR
   Possible causes:
   – The UPIC instance could not be found
   – With UPIC local Unix systems and Windows systems only: the environment
     variable UTMPATH is not set

State change

If the return code is CM_OK, the program enters the “Reset” state. In all other cases, the
program does not change its state.

Notes

● Several CPI-C program runs with the same name can sign on to the UPIC carrier
  system simultaneously.

● A CPI-C program which has been started more than once can also sign on to the same
  UTM application more than once with the same name (e.g. the application name
  assigned to the DEFAULT name). For this purpose, the UTM application must be
  configured as follows:
    – There must be no LTERM partner explicitly generated for this openUTM-Client, i.e.
      no PTERM with its name and PTYPE=UPIC-R must exist for this system in the
      configuration of the UTM application.
    – An LTERM pool (TPOOL) with CONNECT-MODE=MULTI is generated for the
      system on which the client is running. The CPI-C program can then sign on to the
      UTM application under the same name as often as there are LTERM partners
      available in the LTERM pool (the number is set by UTM administration).
● with UPIC local:

To enable the CPI-C program to sign on to the local UTM application, the environment variable UTMPATH must be set.

In rare cases it can occur with local communication that the function terminates with CM_PROGRAM_STATE_CHECK, even though shortly beforehand Disable_UTM_UPIC was called and CM_OK returned. The cause is an incomplete connection shutdown within UTM.

Behavior in the event of errors

CM_PRODUCT_SPECIFIC_ERROR

- The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high; if necessary, reboot your system.

- With UPIC local:
  - Set the UTMPATH environment variable and restart the program.

CM_PROGRAM_STATE_CHECK

Modify program.

CM_PROGRAM_PARAMETER_CHECK

- Modify program.
- Increase the virtual memory if necessary.

Function declaration: Enable_UTM_UPIC

CM_ENTRY Enable_UTM_UPIC ( unsigned char CM_PTR local_name,
  CM_INT32 CM_PTR local_name_length,
  CM_RETURN_CODE CM_PTR return_code)
Extract_Client_Context - Querying the client context

The Extract_Client_Context call provides the program with the client-specific context last sent by openUTM.

The context is buffered by openUTM until the end of the conversation unless it is overwritten with a new context. If the client requests a restart, the context last saved is transferred back to the client together with the last dialog message.

The client context is not saved by openUTM unless the client is signed on with a UTM user ID with restart functionality. This is a requirement for service restart.

The Extract_Client_Context call is permitted in the "Send" and "Receive" state and in the "Reset" state directly after a Receive/Receive_Mapped_Data call.

Extract_Client_Context is not part of the CPI-C specification but is an additional function of the UPIC carrier system.

Syntax

CMECC (conversation_ID, buffer, requested_length, data_received, received_length, return_code)

Parameters

→ conversation_ID Identifier of the conversation already initialized (is supplied by the Initialize call).

← buffer Buffer in which the data is received.

If the value of received_length = 0, the content of buffer is undefined.

→ requested_length Maximum length of the data that can be received.

← data_received Specifies whether the program has received the client context in full.

If the result (return_code) is not CM_OK, the value of data_received is undefined.

data_received can accept the following values.

CM_COMPLETE_DATA_RECEIVED The client context was received in full.

CM_INCOMPLETE_DATA_RECEIVED The client context was not received in full by the program.
← received_length  Length of the received data. If the value of received_length = 0, no client context has been received. The value of received_length is undefined if the result (return_code) is not CM_OK.

← return_code  Result of the function call.

Result (return_code)

CM_OK
The call is OK

CM_CALL_NOT_SUPPORTED
The function is not supported. This return code occurs if no client context can be used because the UTM partner application with Version < 5.0 does not support client context.

CM_PROGRAM_PARAMETER_CHECK
The value in conversation_ID is invalid or the value for requested_length is more than 32767 or less than 1.

The value in conversation_ID is invalid because the function was called more than once after the end of the conversation or because no conversation existed (the Enable_UTM_UPIC call has not yet been followed by an Initialize_Conversation call).

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK
The conversation is not in the "Reset", "Send" or "Receive" state.

Notes

● If a message segment was received with one or more Receive/Receive_Mapped_Data calls (data_received has the value CM_COMPLETE_DATA RECEIVED), the client_context and client_context_length parameters are reset in a subsequent Receive/Receive_Mapped_Data call.

● The value in conversation_ID remains valid for this function call after the end of a conversation until an Initialize_Conversation or an Extract_Client_Context call has been made.

● The internal buffer size is currently limited to 8 bytes.

● openUTM currently always returns a client context with a length of 8 bytes. Consequently, if a valid client context has been received from UPIC, the received_length is 8. If a client context with a length of less than 8 bytes was sent to openUTM, the client context of openUTM is padded with binary zeros to a length of 8 bytes.
If the value for \textit{requested\_length} is less than the length of the internally buffered \textit{client\_context}, the buffer made available by the application program is completely filled and \textit{data\_received} is set to CM\_INCOMPLETE\_DATA\_RECEIVED. If another CMECC call is then immediately made with a sufficiently large value for \textit{requested\_length} (i.e. \(\geq 8\)), the buffer is read in full by such a call.

**Behavior in the event of errors**

CM\_CALL\_NOT\_SUPPORTED
Is not necessarily a program error. If a UPIC-R application communicates with various UTM partner applications, this return code simply means that it is communicating with a UTM application that cannot send a client context (upUTM < V5.0). The program can take note of this return code and dispense with further calls relating to client context.

CM\_PROGRAM\_STATE\_CHECK
Modify program.

CM\_PROGRAM\_PARAMETER\_CHECK
Modify program.

CM\_PRODUCT\_SPECIFIC\_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

**Function declaration: Extract\_Client\_Context**

```c
CM\_ENTRY Extract\_Client\_Context ( 
    unsigned char CM\_PTR conversation\_ID, 
    unsigned char CM\_PTR buffer, 
    CM\_INT32 CM\_PTR requested\_length, 
    CM\_DATA\_RECEIVED\_TYPE CM\_PTR data\_received, 
    CM\_INT32 CM\_PTR received\_length, 
    CM\_RETURN\_CODE CM\_PTR return\_code )
```
**Extract_Communication_Protocol - Querying the PROTOCOL conversation characteristic**

The *Extract_Communication_Protocol* (CMECP) call provides the program with the current value of the *PROTOCOL* conversation characteristic.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

The *Extract_Communication_Protocol* call is permitted only in the "Init" state.

**Syntax**

CMECP (conversation_ID, protocol, return_code)

**Parameters**

→ conversation_ID  
Conversation identifier

← protocol  
This value specifies which communication protocol is used between the UPIC client and the openUTM server.

The following values can be returned for protocol.

CM_COMMUNICATION_PROTOCOL_34  
Communication between the UPIC client and the openUTM server is based on protocol 34.

CM_COMMUNICATION_PROTOCOL_40  
Communication between the UPIC client and the openUTM server is based on protocol 40.

← return_code  
Result of the function call

**Result (return_code)**

CM_OK  
The call is OK

CM_PROGRAM_PARAMETER_CHECK  
The value in conversation_ID is invalid.

CM_PRODUCT_SPECIFIC_ERROR  
The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK  
The conversation is not in the “Initialize” state.
State change

The call does not change the state of the conversation.

Note

If the return code is not CM_OK, the characteristic remains unchanged.

Behavior in the event of errors

CM_PROGRAM_STATE_CHECK
Modify program

CM_PROGRAM_PARAMETER_CHECK
Modify program

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Extract_Communication_Protocol

CM_ENTRY Extract_Communication_Protocol(
    unsigned char               CM_PTR  conversation_ID,
    CM_COMMUNICATION_PROTOCOL   CM_PTR  protocol,
    CM_RETURN_CODE              CM_PTR  return_code )
Extract_Conversation_Encryption_Level

A program uses the Extract_Conversation_Encryption_Level (CMECEL) call to extract the encryption levels which have been set up. The Extract_Conversation_Encryption_Level call is permitted in the following states: “Initialize”, “Send” and “Receive”.

UPIC local: The data transfer is protected by the type of transfer being used. The call Extract_Conversation_Encryption_Level is not supported.

This function belongs to the additional UPIC carrier system functions; it is not a component of the CPI-C interface.

Syntax

CMECEL (conversation_ID, encryption_level, return_code)

Parameters

→ conversation_ID        Conversation identifier
← encryption_level      The following values can be returned:
    CM_ENC_LEVEL_NONE
    The user data of the conversation is transferred in unencrypted form.
    CM_ENC_LEVEL_1
    The user data is encrypted before transfer using the DES algorithm. An RSA key with a key length of 200 bits is used for exchange of the DES key.
    CM_ENC_LEVEL_2
    The user data is encrypted before transfer using the AES algorithm. An RSA key with a key length of 512 bits is used for exchange of the AES key.
    CM_ENC_LEVEL_3
    The user data is encrypted before transfer using the AES algorithm. An RSA key with a key length of 1024 bits is used for exchange of the AES key.
    CM_ENC_LEVEL_4
    The user data is encrypted before transfer using the AES algorithm. An RSA key with a key length of 2048 bits is used for exchange of the AES key.
← return_code            Result of the function call.
Result (*return_code*)

CM_OK
The call is OK.

CM_CALL_NOT_SUPPORTED
The function is not supported. This return code only occurs for UPIC-L. This indicates to the program that encryption is not necessary.

CM_PROGRAM_STATE_CHECK
The conversation is in either the “Start” or the “Reset” state.

CM_PROGRAM_PARAMETER_CHECK
The value of *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_ENCRYPTION_NOT_SUPPORTED
Encryption is not available for this conversation for one of the following reasons:
- the UTM partner application is based on openUTM version < V5.0, or the add-on product openUTM-Crypt is not installed.
- the UTM partner application does not want encryption because the UPIC client is trusted.
- the UPIC client cannot implement encryption because the product openUTM-Client has been installed without the encryption license.

State change
The call does not alter the state of the conversation.

Notes

- CMECEL can only ever supply the current value of the encryption level. The encryption level can always be modified using a subsequent CPI-C call.

- If several conversations are established with the same partner application (or in other words, the communication connection is not set up and cleared down every time), the result of CMECEL will be CMINIT CM_OK after the first call, but after all subsequent CMINIT calls it will be CM_ENCRYPTION_NOT_SUPPORTED. The UPIC library only establishes the connection to the partner application after the first CMALLOC call and thus specifies the encryption option.
**Behavior in event of errors**

- **CM_CALL_NOT_SUPPORTED**
  This is not necessarily an error: If the application is intended for both UPIC-L and UPIC-R this return code just means that the application is linked to a UPIC-L library. If this is the case, encryption is not necessary. The program can take note of this return code and avoid making further calls requesting encryption.

- **CM_PROGRAM_STATE_CHECK**
  Modify program.

- **CM_PROGRAM_PARAMETER_CHECK**
  Modify program.

- **CM_PRODUCT_SPECIFIC_ERROR**
  The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

- **CM_ENCRYPTION_NOT_SUPPORTED**
  This is not necessarily an error: If a UPIC-R application is communicating with several UTM partners some of which implement data encryption and some of which do not, then this return code just means that the UTM application the current application is communicating with either cannot or does not wish to implement encryption. In this case, encryption is not possible. The program can take note of this return code and avoid making further calls requesting encryption.

**Function declaration: Extract_Conversation_Encryption_Level**

```c
Extract_Conversation_Encryption_Level (unsigned char CM_PTR conversation_ID,
                                        CM_ENCRYPTION_LEVEL CM_PTR encryption_level,
                                        CM_RETURN_CODE CM_PTR return_code )
```
Extract_Conversation_State - Querying state of conversation

The Extract_Conversation_State call (CMECS) is used to provide the program with the current state of the conversation.

Syntax

CMECS (conversation_ID, conversation_state, return_code)

Parameters

→ conversation_ID  Conversation identifier
← conversation_state  The value contains the state of the conversation. Values which are valid for UPIC are:
  – CM_INITIALIZE_STATE
  – CM_SEND_STATE
  – CM_RECEIVE_STATE
← return_code  Result of the function call.

Result (return_code)

CM_OK  
The call is OK.
CM_PROGRAM_PARAMETER_CHECK  
The value of conversation_ID is invalid.
CM_PRODUCT_SPECIFIC_ERROR  
The UPIC instance could not be found.

State change

The call does not change the state of the conversation.

Notes

● If the return code is not CM_OK, the value for conversation_state has no significance.
● For the states “Start” and “Reset”, there is never a valid conversation_ID.
Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Extract_Conversation_State

CM_ENTRY Extract_Conversation_State (unsigned char CM_PTR conversation_ID,
                  CM_CONVERSATION_STATE CM_PTR conversation_state,
                  CM_RETURN_CODE CM_PTR return_code )
**Extract_Conversion - Querying the value of the CHARACTER_CONVERSION conversation characteristic**

The *Extract_Conversion* (CMECNV) call provides the program with the current value of the *CHARACTER_CONVERSION* conversation characteristic.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

The *Extract_Conversion* call is permitted only in the “Init” state.

**Syntax**

CMECNV (conversation_ID, character_convertion, return_code)

**Parameters**

→ conversation_ID  Conversation identifier

← character_convertion  The value specifies whether code conversion is carried out or not for the user ID.

The following values can be returned for *character_convertion*.

CM_NO_CHARACTER_CONVERSION  There is no automatic code conversion when data is sent or received.

CM_IMPLICIT_CHARACTER_CONVERSION  Data is automatically converted when sent or received (see also section “Code conversion” on page 104).

← return_code  Result of the function call.

**Result (return_code)**

CM_OK  The call is OK

CM_PROGRAM_PARAMETER_CHECK  The value in conversation_ID is invalid.

CM_PRODUCT_SPECIFIC_ERROR  The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK  The conversation is not in the “Initialize” state.
State change

The call does not change the state of the conversation.

Note

If the return code is not CM_OK, the CHARACTER_CONVERSION characteristic remains unchanged.

Behavior in the event of errors

CM_PROGRAM_STATE_CHECK
Modify program

CM_PROGRAM_PARAMETER_CHECK
Modify program

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Extract_Conversion

CM_ENTRY Extract_Conversion(
  unsigned char conversation_ID,
  CM_CHARACTER_CONVERSION_TYPE conversion_type,
  CM_RETURN_CODE return_code )
**Extract_Cursor_Offset - Querying cursor position offset**

The `Extract_Cursor_Offset` (CMECO) call provides the program with the last value for the cursor position, as sent by openUTM to the client, as long as the cursor is set in the UTM program unit using KDCSCUR.

The `Extract_Cursor_Offset` call is only allowed in the states “Send” and “Receive” and in the “Reset” state after a `Receive-/Receive_Mapped_Data` call.

This function is not a component of the CPI-C specification, it is an additional function of the UPIC carrier system.

**Syntax**

CMECO(conversation_ID, cursor_offset, return_code)

**Parameters**

→ conversation_ID  Conversation identifier
← cursor_offset  Offset of the cursor position.
← return_code  Result of the function call.

**Result (return_code)**

CM_OK  
The call was OK.

CM_CALL_NOT_SUPPORTED  
The function is not supported. This return code occurs if `cursor_offset` cannot be extracted because a version of the UTM partner application that is no longer supported is being used.

CM_PROGRAM_PARAMETER_CHECK  
The value in `conversation_ID` is invalid. The value of `conversation_ID` is invalid because the function was called more than once after terminating the conversation or because no conversation yet exists (after the `Enable_UTM_UPIC` call no `Initialize_Conversation` has been issued).

CM_PRODUCT_SPECIFIC_ERROR  
The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK  
The conversation is not in one of the following states: “Reset”, “Receive” or “Send”.
State change
The call does not change the state of the conversation.

Notes
- If the return code is not CM_OK, the value of cursor_offset has no significance.
- The value for conversation_ID remains valid for this function call, even after terminating a conversation and continues to be valid until Initialize_Conversation or Extract_Cursor_Offset are called.
- A KDCSCUR call overwrites a previous KDCSCUR call in the UTM program unit.
- If an invalid address is entered in KDCSCUR in the UTM program unit, Extract_Cursor_Offset returns the value 0.
- For a +format the address of the attribute field is given as the cursor position.

Behavior in the event of errors
CM_CALL_NOT_SUPPORTED
   Is not necessarily an error: If a UPIC-R application is communicating with several UTM partners, this just means that it is communicating with a UTM application which cannot send a cursor offset (openUTM < V5.0). The program can take note of this return code and avoid making further calls regarding the cursor offset.

CM_PROGRAM_STATE_CHECK
   Modify program

CM_PROGRAM_PARAMETER_CHECK
   Modify program

CM_PRODUCT_SPECIFIC_ERROR
   The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Extract_Cursor_Offset

CM_ENTRY Extrac_Cursor_Offset ( unsigned char CM_PTR conversation_ID,  
                       CM_INT32 CM_PTR cursor_offset,  
                       CM_RETURN_CODE CM_PTR return_code )
Extract_Partner_LU_Name - Querying partner_LU_Name

The Extract_Partner_LU_Name call (CMEPLN) provides the program with the current partner_LU_name of the conversation.

This call belongs to the advanced functions.

Syntax

CMEPLN(conversation_ID, partner_LU_name, partner_LU_name_length, return_code)

Parameters

→ conversation_ID  Conversation identifier
← partner_LU_name  Returns the partner_LU_name. The length of the parameter must be at least 32 bytes.
← partner_LU_name_length  Specifies the length of the value returned in partner_LU_name. Minimum: 1, maximum: 32.
← return_code  Result of the function call.

Result (return_code)

CM_OK  The call is OK.
CM_PROGRAM_PARAMETER_CHECK  The value in conversation_ID is invalid.
CM_PRODUCT_SPECIFIC_ERROR  The UPIC instance could not be found.
CM_PROGRAM_STATE_CHECK  The conversation is not in the “Initialize” state.

State change

The call does not change the state of the conversation.

Note

If the return code is not CM_OK, the value of partner_LU_name has no significance.
Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

CM_PROGRAM_STATE_CHECK
Modify program

Function declaration: Extract_Partner_LU_Name

CM_ENTRY Extract_Partner_LU_Name (unsigned char CM_PTR conversation_ID,
    unsigned char CM_PTR partner_LU_name,
    CM_INT32 CM_PTR partner_LU_name_length,
    CM_RETURN_CODE CM_PTR return_code)
Extract_Secondary_Information - Querying secondary information

The Extract_Secondary_Information (CMESI) call provides the program with expanded information (secondary information) relating to the return code of the most recent CPI-C call.

Syntax

CMESI (conversation_ID, call_ID, buffer, requested_length, data_received, received_length, return_code)

Parameters

→ conversation_ID Identifier for the started conversation (supplied by the Initialize call).
→ call_ID Specifies the function on which secondary information is required.
← buffer Buffer which receives the data. If the return code of data_received is CM_NO_DATA_RECEIVED, the contents of buffer are undefined.
→ requested_length Maximum length of data that can be received.
← data_received Specifies whether the program has completely received the secondary information. If the result (return_code) is not CM_OK, the value of data_received is undefined.

data_received can have one of the following values:

– CM_COMPLETE_DATA_RECEIVED
  The secondary information was received completely.

– CM_INCOMPLETE_DATA_RECEIVED
  The secondary information was incompletely received by the program.

← received_length Length of received data. The value of received_length is undefined as long as the result (return_code) does not have the value CM_OK.
← return_code Result of the function call.
Result (return_code)

CM_OK
The call is OK

CM_NO_SECONDARY_INFORMATION
There is no secondary information available for the call of the specified conversation.

CM_PROGRAM_PARAMETER_CHECK
The value of conversation_ID is invalid, the call_ID specifies CMESI or an invalid value, or the value of requested_length is greater than 32767 or less than 1.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

Notes

- The program should make this call immediately after receiving a return_code. Subsequent CPI-C calls can overwrite the secondary information. If there is no conversation, for example, if the library is in the “Reset” state, then conversation_ID is ignored.

- When the Extract_Secondary_Information call is successfully terminated, the returned secondary information does not remain saved. The same information will no longer be available in a subsequent Extract_Secondary_Information call.

- The program cannot use the call to extract secondary information from a previous Extract_Secondary_Information call.

- The full complexity of this function is not implemented as laid down in the CPI-C specification. The simplifications in comparison with CPI-C are as follows:
  - The internal buffer is limited to a size of 1024 bytes.
  - If the value of requested_length is less than the length of the secondary information saved internally, the buffer made available by the application program is filled completely and data_received is set to CM_INCOMPLETE_DATA_RECEIVED. It is not possible to obtain the remaining data using further CMESI calls.

Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.
Function declaration: Extract_Secondary_Information

CM_ENTRY Extract_Secondary_Information (unsigned char CM_PTR conversation_ID,
CM_INT32 CM_PTR call_ID,
unsigned char CM_PTR buffer,
CM_INT32 CM_PTR requested_length,
CM_DATA_RECEIVED_TYPE CM_PTR data_received,
CM_INT32 CM_PTR received_length,
CM_RETURN_CODE CM_PTR return_code)
Extract_Secondary_Return_Code - Querying secondary return codes

The Extract_Secondary_Return_Code (CMESRC) call provides the program with secondary return codes that relate to the primary return code of the last CPI-C call.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

Syntax

CMESRC (conversation_ID, call_ID, secondary_return_code, return_code)

Parameters

→ conversation_ID  Identifier of the conversation already initialized (is supplied by the Initialize call).
→ call_ID  Specifies the function whose secondary return code is to be output.
← secondary_return_code  Supplies the secondary return code of the last CPI-C call. If the result is not CM_OK, the value of secondary_return_code is undefined.
← return_code  Result of the function call.

Result (return_code)

CM_OK  The call is OK
CM_NO_SECONDARY_RETURN_CODE  There is no secondary return code for the call of the specified conversation.
CM_PROGRAM_PARAMETER_CHECK  The value in conversation_ID is invalid, the call_ID specifies CMESRC or an invalid value.
CM_PRODUCT_SPECIFIC_ERROR  The UPIC instance could not be found.
**Secondary return code** *(secondary_return_code)*

**CM_SECURITY_USER_UNKNOWN**  
The specified user ID is not generated.

**CM_SECURITY_STA_OFF**  
The specified user ID is locked by generation or administration.  
The administrator of the UTM application can remove the lock.

**CM_SECURITY_USER_IS_WORKING**  
Somebody has already signed on to this UTM application with this user ID.

**CM_SECURITY_OLD_PSWORD_WRONG**  
The old password entered is incorrect.

**CM_SECURITY_NEW_PSWORD_WRONG**  
The new password information cannot be used. Possible cause: minimum period of validity not yet expired.  
Use the old password until its validity expires.

**CM_SECURITY_NO_CARD_READER**  
The user is generated with a magnetic stripe card and cannot sign on via UPIC.

**CM_SECURITY_CARD_INFO_WRONG**  
The user is generated with a chipcard and cannot sign on via UPIC.

**CM_SECURITY_NO_RESOURCES**  
Sign-on is not possible at the moment. Possible cause:  
– a resource bottleneck, or  
– the maximum number of simultaneous users signed on has been reached (see KDCDEF statement MAX CONN-USERS=), or  
– an inverse KDCDEF is running  
Try again later.

**CM_SECURITY_NO_KERBEROS_SUPPORT**  
The user is generated with a Kerberos principal and cannot sign on via UPIC.

**CM_SECURITY_TAC_KEY_MISSING**  
The current LTERM is not authorized to resume the service.

**CM_SECURITY_PWD_EXPIRED_NO_RETRY**  
The validity period of the user password has expired, the UTM application is generated with SIGNON GRACE=NO.  
The client user can no longer sign on. He or she must request the administrator of the UTM application to issue a new password.
CM_SECURITY_COMPLEXITY_ERROR
The new password is not sufficiently complex. See KDCDEF control statement USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT
The new password is too short.
See KDCDEF control statement USER PROTECT-PW=.

CM_SECURITY_UPD_PSWORD_WRONG
The password transferred by KDCUPD does not satisfy the complexity or minimum length requirement defined in application generation.
See KDCDEF control statement USER PROTECT-PW=.

The password must be changed by administration before the user can sign on again.

CM_SECURITY_TA_RECOVERY
A transaction restart is required for the specified user ID.

CM_SECURITY_PROTOCOL_CHANGED
The user has an open service that cannot be resumed from a UPIC client.

CM_SECURITY_SHUT_WARN
The application run is terminated, only users with administration authorization may still sign on.

Sign on is not possible until the UTM application has been restarted.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
The encryption mechanism required to resume the open service is not available on the connection.

CM_SECURITY_PWD_EXPIRED_RETRY
The validity period of the user password has expired, the UTM application is generated with SIGNON GRACE=YES.

The client can nevertheless sign on by entering a suitable new password in addition to the old password.

If the new password is the same as the old password, openUTM rejects sign-on. When working with openUTM > 5.1A30, the secondary return code set by UPIC in this case is CM_SECURITY_NEW_PSWORD_WRONG.
The following secondary return codes only occur in the context of UTM cluster applications:

**CM_SECURITY_USER_GLOBALY_UNKNOWN**
The specified user ID is not recognized in the cluster user file.

**CM_SECURITY_USER_SIGNED_ON_OTHER_NODE**
A user has already signed on to another node application with this user ID.

**CM_SECURITY_TRANSIENT_ERROR**
A temporary error occurred during signon. The cluster user file could not be accessed in the time configured in the node application.

Try signing on again later.

**Notes**

- The program should issue this call immediately after receipt of a return code. Subsequent CPI-C calls may overwrite the secondary return code. The \textit{conversation\_ID} is ignored if no conversation exists, i.e. the library is in the "Reset" state.
- If the \textit{Extract\_Secondary\_Return\_Code} call terminates successfully, the secondary return code supplied is no longer saved. The same return code is then no longer available in the next \textit{Extract\_Secondary\_Return\_Code} call.
- The program cannot use the call to obtain a secondary return code from a preceding \textit{Extract\_Secondary\_Return\_Code} call.
- The secondary return code and associated description can be found in the individual UPIC calls.

**State change**

No state change.

**Behavior in the event of errors**

**CM\_PROGRAM\_PARAMETER\_CHECK**
Modify program.

**CM\_PRODUCT\_SPECIFIC\_ERROR**
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.
Function declaration: Extract_Secondary_Return_Code

CM_ENTRY Extract_Secondary_Return_Code ( 
    unsigned char CM_PTR  conversation_ID, 
    CM_INT32      CM_PTR  call_ID, 
    CM_RETURN_CODE CM_PTR  secondary_return_code, 
    CM_RETURN_CODE CM_PTR  return_code )
**Extract_Transaction_State - Querying service and transaction state of the server**

The `Extract_Transaction_State` call provides the program with the service and transaction state sent to the client by openUTM.

The `Extract_Transaction_State` call is permitted only in the "Send" and "Receive" state and in the "Reset" state directly after a `Receive/Receive_Mapped_Data` call.

This function is not a component of the CPI-C specification but is an additional function of the UPIC carrier system.

**Syntax**

CMETS (conversation_ID, transaction_state, requested_length, transaction_state_length, return_code)

**Parameters**

- → conversation_ID          Conversation identifier
- ← transaction_state        Transaction and service state
- → requested_length          Maximum length of the data that can be received
- ← transaction_state_length  Length of the message received
- ← return_code               Result of the function call.

**Result (return_code)**

CM_OK
  The call is OK

CM_CALL_NOT_SUPPORTED
  The function is not supported. This return code occurs if no transaction_state can be received because the UTM partner application with Version < 5.0 does not support this function.

CM_PROGRAM_PARAMETER_CHECK
  The value in conversation_ID is invalid.
  The value in conversation_ID is invalid because the function was called more than once after the end of the conversation or because no conversation existed (the `Enable_UTM_UPIC` call has not yet been followed by an `Initialize_Conversation` call).
CM_PRODUCT_SPECIFIC_ERROR
   The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK
   The conversation is not in the "Reset", "Send" or "Receive" state.

State change
The call does not change the state of the conversation.

Notes
● If the return code is not CM_OK, the value of transaction_state has no significance.
● The value of conversation_ID remains valid for this function call after the end of a conversation until an Initialize_Conversation or an Extract_Transaction_State call has been made.
● If the value of transaction_state_length is 0, no new transaction_state was received.

Behavior in the event of errors
CM_CALL_NOT_SUPPORTED
   Is not necessarily a program error. If a UPIC-R application communicates with various UTM partners, this return code simply means that it is communicating with a UTM partner application that cannot send the transaction and service state (openUTM < V5.0). The program can take note of this return code and dispense with further Extract_Transaction_State calls.

CM_PROGRAM_STATE_CHECK
   Modify program.

CM_PROGRAM_PARAMETER_CHECK
   Modify program.

CM_PRODUCT_SPECIFIC_ERROR
   The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.
Description of *transaction_state*

The first two bytes of *transaction_state* contain the information on the service and transaction state of the server and can be evaluated accordingly. The remaining bytes (dd dd) contain internal diagnostics information.

<table>
<thead>
<tr>
<th>transaction_state (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 08 dd dd</td>
<td>End of the processing step; the transaction is not completed, the service is still open (PEND/PGWT KP).</td>
</tr>
<tr>
<td>18 08 dd dd</td>
<td>End of the processing step; the transaction is completed, the service is still open (PEND RE/PGWT CM).</td>
</tr>
<tr>
<td>1A 04 dd dd</td>
<td>End of a service and end of the transaction (PEND FI).</td>
</tr>
<tr>
<td>30 04 dd dd</td>
<td>End of a service with memory dump (PEND ER).</td>
</tr>
<tr>
<td>31 04 dd dd</td>
<td>End of a service (system PEND ER, i.e. PEND ER by openUTM).</td>
</tr>
<tr>
<td>32 04 dd dd</td>
<td>End of a service due to abnormal task termination (only openUTM on BS2000/OSD).</td>
</tr>
<tr>
<td>20 04 dd dd</td>
<td>Roll back of the first transaction of a service and end of the service (PEND RS).</td>
</tr>
<tr>
<td>21 04 dd dd</td>
<td>Roll back of a follow-up transaction to the last synchronization point; the service is still open (PEND RS).</td>
</tr>
</tbody>
</table>

For further information on PEND and PGWT calls refer to the openUTM manual "Programming Applications with KDCS".

**Function declaration: Extract_Transaction_State**

```c
CM_ENTRY Extract_Transaction_State(
    unsigned char      CM_PTR  conversation_ID,
    unsigned char      CM_PTR  transaction_state,
    CM_INT32           CM_PTR  requested_length,
    CM_INT32           CM_PTR  transaction_state_length,
    CM_RETURN_CODE     CM_PTR  return_code )
```
Initialize_Conversation - Initializing the conversation characteristics

The Initialize_Conversation (CMINIT) call reads the entry specified by the symbolic destination name in the upicfile and initializes the conversation characteristics. The characteristics partner_LU_name, partner_LU_name_lth, TP_name, and TP_name_length are assigned corresponding values from the upicfile. All other conversation characteristics are initialized with default values.

In addition to initializing the conversation characteristics, this call also specifies whether the user data will be converted automatically from ASCII to EBCDIC (or vice versa) during the next Send or Receive calls. Conversion takes place:

- in Unix systems and Windows systems, if the identifier HD is placed before the symbolic destination name
- in BS2000/OSD, if the identifier SD is placed before the symbolic destination name.

For details see also page 324.

The call returns an eight-character conversation_ID. This uniquely identifies the conversation and must be used in all subsequent CPI-C calls to address the conversation.

It is possible to change the initial values of the conversation characteristics TP_name, TP_name_length, receive_type and deallocate_type at a later stage. The Set_TP_Name, Set_Receive_Type and Set_Deallocate_Type calls are provided for this purpose. A value changed with a Set call is applicable until the end of the conversation or until a new Set call is issued.

The Set calls are not part of the CPI-C starter set, but are advanced-function calls.

Syntax

CMINIT (conversation_ID, sym_dest_name, return_code)
Parameters

← conversation_ID  Identifier assigned to the conversation and returned to the program as a result parameter.

→ sym_dest_name  If you use no upicfile, you must specify 8 blanks for sym_dest_name ("empty sym_dest_name").
If you work with the upicfile, enter the reference to the side information (8-character name). For sym_dest_name you can also specify 8 blanks ("empty sym_dest_name").
In this case the symbolic destination name .DEFAULT is sought in the side information (see page 331) and the corresponding values are set for partner_LU_name, partner_LU_name_lth, TP_name and TP_name_length. If you are working with the upicfile, you can specify 8 blanks for sym_dest_name ("empty sym_dest_name").

← return_code  Result of the function call.

Result (return_code)

CM_OK  The call is OK.

CM_PROGRAM_PARAMETER_CHECK
  – The value of sym_dest_name or local_name (with Enable_UTM_UPIC) is invalid or the specified entry in the upicfile could not be read or is syntactically invalid.
  – An attempt (if any) to sign on to or sign off from the transport interface was unsuccessful.
  – In sym_dest_name or in local_name (with Enable_UTM_UPIC) an empty name was specified but there is no corresponding default entry in the upicfile or the default entry is invalid.
  – Error in the upicfile:
     The CD entries for the specified sym_dest_name are not consecutive or the CD entries for the specified sym_dest_name contain different TACs.

CM_PRODUCT_SPECIFIC_ERROR
  – A conversation is already active for this program, or no Enable_UTM_UPIC call has been issued yet.
  – The transport interface did not respond as expected.

State change

If the return code is CM_OK, the program enters the “Initialize” state and the conversation characteristics are initialized. Further details can be found in “Conversation characteristics” on page 87. In all other error conditions, the program does not change its state.
Initialize_Conversation

CPI-C calls in UPIC

Notes

- The Initialize_Conversation call must be executed by the program before another call is issued for this conversation.

- If the Initialize_Conversation call or the subsequent Set calls of the program supply invalid information for establishing the conversation, errors of a syntactical kind are detected immediately but semantic errors are not detected until the Allocate (CMALLC) call is executed.

- Several programs can sign on under the same name if CONNECT-MODE=MULTI is defined for the corresponding TPOOL statement.

- With a remote connection:
  - The function may sign the program on to the transport system (e.g. TCP/IP, PCMX, CMX, BCAM) using the name of the preceding Enable_UTM_UPIC call. No signing on takes place if the program is already signed on with the same name.
  - Any remaining connection to a partner (except for the partner in the upicfile) is shut down.

- With a local connection (UPIC on Unix systems and Windows systems):
  - The function performs the sign-on to the UTM-internal process communication (with the UTM application name from the upicfile) if the program is not yet signed on with the same name. If the program is still signed on with a different name, it is first signed off from the UTM-internal process communication. An existing conversation with this UTM application is hereby implicitly shut down. Only then is the program signed on with the new name.
  - At sign-on to the UTM application, the applifile of the UTM application is read. For this purpose the shell variable UTMPATH, which points to the corresponding UTM directory utmpath, is interpreted. This variable must have been set.
Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
- Create the upicfile or set the environment variable or job variable UPICPATH to the correct values. Check the TNS entry or the BCMAP entry in BS2000/OSD.
- Enter the current sym_dest_name in the upicfile or check the entry for sym_dest_name for correct syntax.
- With a local connection: set the environment variable UTMPATH to the correct values. It is also possible that there is no longer a semaphore available.
- Modify the upicfile: Check and adjust the CD entries.

CM_PRODUCT_SPECIFIC_ERROR
Modify the program or inform the service department and produce diagnostic report.

Function declaration: Initialize_Conversation

CM_ENTRY Initialize_Conversation ( unsigned char CM_PTR conversation_ID,
                                unsigned char CM_PTR sym_dest_name,
                                CM_RETURN_CODE CM_PTR return_code)
Prepare_To_Receive - Changing state from “Send” to “Receive”

The Prepare_To_Receive (CMPTR) call has the following effect:

– All data which is still stored in the local send buffer at the time of the call is transferred to the UTM service together with permission to send.

– Once the data has been transferred from the send buffer to the UTM service, the conversation switches from the “Send” state to the “Receive” state.

Prepare_To_Receive can only be called when the conversation is in the “Send” state, but not directly after the Allocate call or after receipt of permission to send from the partner. In these two exceptional cases, a Send_Data or Send_Mapped_Data call must be issued before the Prepare_To_Receive call.

After the Prepare_To_Receive call, a Receive or Receive_Mapped_Data call must be issued. Before the Receive or Receive_Mapped_Data call, however, Set_Receive_Timer or Set_Receive_Type may be called.

Syntax

CMPTR (conversation_ID, return_code)

Parameters

→ conversation_ID  Identifier of the conversation
← return_code  Result of the function call
Result (*return_code*)

CM_OK
The call is OK. The conversation has switched from the “Send” state to the “Receive” state.

CM_DEALLOCATED_ABEND
Possible causes:
- abnormal termination of the UTM service
- termination of the UTM application
- connection shutdown by UTM administration
- connection shutdown by the transport system
- Connection shutdown by openUTM because the maximum permitted number of users (MAX statement, CONN-USER=) has been exceeded. This may also occur if an administrator user was transferred in the `Set_Conversation_Security_User_ID` call. This is the case if a user ID that has no administration authorization is assigned to the LTERM partner of the CPI-C program in the UTM application (via LTERM ...USER=).

The program enters the “Reset” state.

CM_PRODUCT_SPECIFIC_ERROR
Possible causes:
- The UPIC instance could not be found.
- The `Prepare_To_Receive` call was issued immediately after an `Allocate` call instead of a `Send_Data` or `Send_Mapped_Data` call.

CM_PROGRAM_STATE_CHECK
The call is not permitted in the current state of the conversation.

CM_PROGRAM_PARAMETER_CHECK
The value of `conversation_ID` is invalid.

CM_Resource_FAILURE_NO_RETRY
An error has occurred which led to a premature termination of the conversation (e.g. a protocol error or a premature loss of the network connection). The program enters the “Reset” state.

State change
- If the result of the call is CM_OK, the state of the conversation changes from “Send” to “Receive”.
- With the following results, the program enters the “Reset” state:
  CM_DEALLOCATED_ABEND
  CM_RESOURCE_FAILURE_NO_RETRY
- In all other error conditions, the program does not change its state.
Behavior in the event of errors

CM_PRODUCT_SPECIFIC_ERROR
- Modify program.
- The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CMRESOURCE_FAILURE_NO_RETRY
Inform the service department and produce a diagnostic report. A fault in the transport system could also be the reason for this error code.

Function declaration: Prepare_To_Receive

CM_ENTRY Prepare_To_Receive (unsigned char CM_PTR conversation_ID, 
CM_RETURN_CODE CM_PTR return_code )
Receive - Receiving data from a UTM service

A program uses the *Receive* (CMRCV) call to receive information from a UTM service. The call can be executed with or without blocking.

- The *Receive* call is "blocking" when the *receive_type* characteristic has the value CM_RECEIVE_AND_WAIT. If no information (data or permission to send) is present at the time of the *Receive* call, the program run waits in the *Receive* until information is available for this conversation. Only then does the program run return from the *Receive* call and bring back the information. If there is information available at the time of the call, the program receives it without waiting.

To limit the wait time for a blocking *Receive* call, appropriate timers should be set in the UTM partner application.

- The *Receive* call is "non-blocking" when the *receive_type* characteristic has the value CM_RECEIVE_IMMEDIATE. If no information is present at the time of the *Receive* call, the program run does not wait until information for this conversation arrives. The program run returns from the *Receive* call immediately. If there is already information available, it is transferred to the program.

**UPIC local:**

Local connection via UPIC local does not support the non-blocking *Receive* call.

You can set the *receive_type* characteristic with the *Set_Receive_Type* call before the *Receive* call. After a conversation has been initialized, the blocking receive is set by default.

**Syntax**

CMRCV (conversation_ID, buffer, requested_length, data_received, received_length, status_received, control_information_received, return_code)

**Parameters**

→ conversation_ID  Identifier of the conversation.

← buffer  Buffer in which the data is received. If the return value of *data_received* is CM_NO_DATA_RECEIVED, the contents of *buffer* are undefined.

→ requested_length  Maximum length of data that can be received.
received

规定的程序已经收到数据。

如果结果（return_code）既不是CM_OK也不是
CM_DEALLOCATED_NORMAL，则data_received的值
未定义。

data_received可以有一个以下的值：

CM_NO_DATA_RECEIVED
没有数据可供程序使用。发送权限可能已经
收到。

CM_COMPLETE_DATA_RECEIVED
一个完整的消息已经可供程序接收。

CM_INCOMPLETE_DATA_RECEIVED
一个消息未完全传输到程序。如果data_received
具有此值，程序必须重复调用Receive调用
直到消息完全接收，即data_received具有
值CM_COMPLETE_DATA_RECEIVED。

← received_length
长度的数据接收。

如果程序没有接收到数据（data_received=CM_NO_DATA_RECEIVED）或如果结果不
是CM_OK或CM_DEALLOCATE_NORMAL，则received_length的值
未定义。

← status_received
规定程序是否收到发送权限。

status_received可以有一个以下的值：

CM_NO_STATUS_RECEIVED
发送权限未得到。

CM_SEND_RECEIVED
UTM服务已经将发送权限传递给程序。

程序必须随后调用Send_Data调用。

除非返回码是CM_OK，status_received的值
未定义。

← control_information_received

这仅支持语义上，并始终具有值
CM_REQ_TO_SEND_NOT_RECEIVED。

如果返回码不是CM_OK或CM_DEALLOCATE_NORMAL，
control_information_received的值
未定义。

← return_code
结果的函数调用。

openUTM-Client for the UPIC Carrier System
**Result** *(return_code)*

**CM_OK**

If the return code is CM_OK, the program has one of the following states after function call:

- “Receive”, if the value of `status_received` is CM_NO_STATUS_RECEIVED.
- “Send”, if the value of `status_received` is CM_SEND_RECEIVED.

**CM_SECURITY_NOT_VALID**

Possible causes:
- an invalid UTM user ID in the `Set_Conversation_Security_User_ID` call
- an invalid password in the `Set_Conversation_Security_Password` call
- the UTM application was generated without USER
- the user cannot sign on to the UTM application due to a resource bottleneck

If the UPIC application communicates with an openUTM application that returns a detailed result of the authorization check (§ V5.1A30), the UPIC library supplies a secondary return code that describes the cause in detail. The results received by the program are listed under `secondary_return_code`, see page 189.

The secondary return codes can also be queried using the `Extract_Secondary_Return_Code` call, see page 170.

**CM_TPN_NOT_RECOGNIZED**

Possible causes:
- invalid transaction code (TAC) in the `upicfile` or in the `Set_TP_Name` call, e.g.:
  - the TAC is not generated
  - you are not authorized to call this TAC
  - the TAC is permitted only as a follow-up TAC
  - the TAC is not a dialog TAC
  - TAC is generated with encryption, but user data is sent without implementing encryption, or encryption is not supported for the connection, or the encrypted data does not have the required encryption level.
- a service restart with KDCDISP was rejected as no UTM user ID generated with `RESTART=YES` was specified

**CM_TP_NOT_AVAILABLE_NO_RETRY**

A service restart with KDCDISP is not possible as the UTM application has been regenerated.

**CM_TP_NOT_AVAILABLE_RETRY**

A service restart was rejected as the UTM application has been terminated.
CM_DEALLOCATED_ABEND
Possible causes:
– abnormal termination of the UTM service
– termination of the UTM application
– connection shutdown by UTM administration
– connection shutdown by the transport system
– connection shutdown by UTM because the maximum permitted number of users (MAX statement, CONN-USERS=) has been exceeded. This may also occur if an administrator user was transferred in the Set_Conversation_Security_User_ID call but the user ID implicitly assigned to the connection by UTM generation or the (connection) user ID explicitly assigned using the statement LTERM..., USER= is not an administrator user (CONN-USERS applies only for users without administration authorization).

The program enters the “Reset” state.

CM_DEALLOCATED_NORMAL
A PEND-FI call was executed in the UTM service. The program enters the state “Reset”.

CM_RESOURCE_FAILURE_RETRY
A temporary resource bottleneck led to termination of the conversation. It may not be possible to buffer any further data in the UTM page pool. If the error recurs, the page pool of the UTM application should be enlarged (MAX statement, PGPOOL=).

CM_RESOURCE_FAILURE_NO_RETRY
An error occurred which led to premature termination of the conversation (e.g. protocol error or premature loss of network connection).

CM_PROGRAM_STATE_CHECK
The call is not permitted in the current state. The contents of all other variables are undefined.

CM_PROGRAM_PARAMETER_CHECK
The conversation_ID is invalid or the value in requested_length is greater than 32767 or less than 0. The contents of all other variables are undefined.

CM_PRODUCT_SPECIFIC_ERROR
A Receive call was issued instead of a Send_Data call (only directly after an Allocate call).

CM_OPERATION_INCOMPLETE
The Receive call was interrupted by the expiry of the timer that was set with Set_Receive_Timer. No data was received.

CM_UNSUCCESSFUL
receive_type has the value CM_RECEIVE_IMMEDIATE and there is currently no data available for the conversation.
**Secondary return code** *(secondary_return_code)*

CM_SECURITY_USER_UNKNOWN
The specified user ID is not generated.

CM_SECURITY_STA_OFF
The specified user ID is locked.

CM_SECURITY_USER_IS_WORKING
Another user is already signed on with this user ID.

CM_SECURITY_OLD_PSWORD_WRONG
The old password entered is incorrect.

CM_SECURITY_NEW_PSWORD_WRONG
The new password information cannot be used. Possible cause: minimum period of validity not yet expired.

CM_SECURITY_NO_CARD_READER
The user is generated with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG
The user is generated with a chipcard and cannot sign on via UPIC.

CM_SECURITY_NO_RESOURCES
Sign-on is not possible at the moment. Possible cause:
- a resource bottleneck, or
- the maximum number of simultaneous users signed on has been reached (see KDCDEF statement MAX CONN-USERS=), or
  - an inverse KDCDEF is running

Try again later.

CM_SECURITY_NO_KERBEROS_SUPPORT
The user is generated with a Kerberos principal and cannot sign on via UPIC.

CM_SECURITY_TAC_KEY_MISSING
The current LTERM is not authorized to resume the service.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
The validity period of the user password has expired.

CM_SECURITY_COMPLEXITY_ERROR
The new password is not sufficiently complex.

CM_SECURITY_PASSWORD_TOO_SHORT
The new password is too short.
CM_SECURITY_UPD_PSWORD_WRONG
The password transferred by KDCUPD does not satisfy the complexity or minimum
length requirement defined in application generation.

CM_SECURITY_TA_RECOVERY
A transaction restart is required for the specified user ID.

CM_SECURITY_PROTOCOL_CHANGED
The open service cannot be resumed from this LTERM partner.

CM_SECURITY_SHUT_WARN
The administrator has issued a SHUT WARN. Normal users may no longer sign on
to the UTM application, only the administrator may still sign on.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
The encryption mechanism required to resume the open service is not available on
the connection.

CM_SECURITY_PWD_EXPIRED_RETRY
The validity period of the user password has expired.

The following secondary return codes only occur in the context of UTM cluster applications:

CM_SECURITY_USER_GLOBALY_UNKNOWN
The specified user ID is not recognized in the cluster user file.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
A user has already signed on to another node application with this user ID.

CM_SECURITY_TRANSIENT_ERROR
A temporary error occurred during signon. The cluster user file could not be
accessed in the time configured in the node application.

Try signing on again later.
State change

- If the return code is CM_OK, the program has one of the following states after function call:
  “Receive” if the value of status_received is CM_NO_STATUS_RECEIVED.
  “Send” if the value of status_received is CM_SEND_RECEIVED.

- With the following return codes, the program enters the “Reset” state:
  - CM_DEALLOCATED_ABEND
  - CM_DEALLOCATED_NORMAL
  - CM_SECURITY_NOT_VALID
  - CM_TPN_NOT_RECOGNIZED
  - CM_TPN_NOT_AVAILABLE_RETRY/NO_RETRY
  - CM_RESOURCE_FAILURE_RETRY/NO_RETRY
  - CM_SECURITY_USER_UNKNOWN
  - CM_SECURITY_STA_OFF
  - CM_SECURITY_USER_IS_WORKING
  - CM_SECURITY_OLD_PSWORD_WRONG
  - CM_SECURITY_NEW_PSWORD_WRONG
  - CM_SECURITY_NO_CARD_READER
  - CM_SECURITY_CARD_INFO_WRONG
  - CM_SECURITY_NO_RESOURCES
  - CM_SECURITY_NO_KERBEROS_SUPPORT
  - CM_SECURITY_TAC_KEY_MISSING
  - CM_SECURITY_PWD_EXPIRED_NO_RETRY
  - CM_SECURITY_COMPLEXITY_ERROR
  - CM_SECURITY_PASSWORD_TOO_SHORT
  - CM_SECURITY_UPD_PSWORD_WRONG
  - CM_SECURITY_TA_RECOVERY
  - CM_SECURITY_PROTOCOL_CHANGED
  - CM_SECURITY_SHUT_WARN
  - CM_SECURITY_ENC_LEVEL_TOO_HIGH
  - CM_SECURITY_PWD_EXPIRED_RETRY
  - CM_SECURITY_PWD_EXPIRED_RETRY
  - CM_SECURITY_USER_GLOBALY_UNKNOWN
  - CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
  - CM_SECURITY_TRANSIENT_ERROR

- In all other error conditions, the program does not change its state.
Notes

- If a maximum wait time was set with the `Set_Receive_Timer` call before a blocking `Receive` call, the program run returns from the `Receive` call at the latest once the wait time has expired, and the `Receive` call then returns the result (`return_code`) `CM_OPERATION_INCOMPLETE`.

- With a `Receive` call, a program can only receive the amount of data specified in the `requested_length` parameter. It is therefore possible that a message is only partially received with the `Receive` call. The `data_received` parameter indicates as shown below whether a complete message available for the program was received:
  - If the program has already received the complete message, the `data_received` parameter has the value `CM_COMPLETE_DATA_RECEIVED`.
  - If the program has not yet received all data of the message, the `data_received` parameter has the value `CM_INCOMPLETE_DATA_RECEIVED`. The program must then continue to call `Receive` until `data_received` has the value `CM_COMPLETE_DATA_RECEIVED`.

- A program can use a single call to receive both data and permission to send. The `return_code`, `data_received`, and `status_received` parameters supply details on the kind of information received by a program.

- If the program issues the `Receive` call in the “Send” state, permission to send is passed to the UTM service. The send direction of the conversation is thus changed.

- A `Receive` call with `requested_length = 0` has no special meaning. If data is available, it is received in the length 0 and `data_received = CM_INCOMPLETE_DATA_RECEIVED`. If no data is available, permission to send can be received. This means that either data or permission to send can be received, but not both.

- If the UTM partner application transfers a format identifier (structure information concerning the transferred file), this will be received by UPIC (no error occurs in the UTM service), but it cannot be passed on to the program. Data together with format IDs can only be read with `Receive_Mapped_Data`. 
Behavior in the event of errors

CMRESOURCE_FAILURE_RETRY
Re-establish conversation.

CMRESOURCE_FAILURE_NO_RETRY
Notify the service department and produce diagnostic report.
A fault in the transport system can also cause this return code.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
Modify program.

CM_SECURITY_USER_UNKNOWN
The UTM user ID is not generated. Use a user ID that is generated or generate or
dynamically configure the user ID you want.

CM_SECURITY_STA_OFF
Generate the user ID with STATUS=ON or unlock it using administration facilities.

CM_SECURITY_USER_IS_WORKING
Use another UTM user ID or terminate the service of the user already signed on.

CM_SECURITY_OLD_PSWORD_WRONG
Enter the password correctly.

CM_SECURITY_NEW_PSWORD_WRONG
Use the old password until its validity expires.

CM_SECURITY_NO_CARD_READER
The user is generated with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG
The user is generated with a chipcard.

CM_SECURITY_NO_RESOURCES
Try again later.

CM_SECURITY_NO_KERBEROS_SUPPORT
The user is generated with a Kerberos principal and cannot sign on via UPIC.

CM_SECURITY_TAC_KEY_MISSING
Generation or modify program.
CM_SECURITY_PWD_EXPIRED_NO_RETRY
The validity period of the password has expired. The password must be changed using administration facilities before the user can sign on again.

CM_SECURITY_COMPLEXITY_ERROR
Select a new password that satisfies the requirements of the generated complexity level, see KDCDEF statement USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT
Select a longer password or change generation, see KDCDEF statement USER PROTECT-PW= length, ... (value for the minimum length).

CM_SECURITY_UPD_PSWORD_WRONG
The password is not sufficiently complex or is too short, see KDCDEF statement USER PROTECT-PW=. The password must be changed using administration facilities before the user can sign on again.

CM_SECURITY_TA_RECOVERY
A transaction restart is required for the specified user ID.

CM_SECURITY_PROTOCOL_CHANGED
The user has an open service that cannot be resumed from a UPIC client.

CM_SECURITY_SHUT_WARN
The UTM application is terminated: only users with administration authorization may sign on. Wait until the application has been restarted.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
The encryption mechanism required to resume the open service is not available on the connection.

CM_SECURITY_PWD_EXPIRED_RETRY
Repeat initiation of the conversation specifying the old password and the new password.

The following secondary return codes only occur in the context of UTM cluster applications:

CM_SECURITY_USER_GLOBALY_UNKNOWN
The specified user ID is not recognized in the cluster user file.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
A user has already signed on to another node application with this user ID.

CM_SECURITY_TRANSIENT_ERROR
A temporary error occurred during signon. The cluster user file could not be accessed in the time configured in the node application.

Try signing on again later.
Function declaration: Receive

CM_ENTRY Receive ( unsigned char CM_PTR conversation_ID,/
        unsigned char CM_PTR buffer,
        CM_INT32 CM_PTR requested_length,
        CM_DATA_RECEIVED_TYPE CM_PTR data_received,
        CM_INT32 CM_PTR received_length,
        CM_STATUS_RECEIVED CM_PTR status_received,
        CM_CONTROL_INFORMATION_RECEIVED CM_PTR control_information_received,
        CM_RETURN_CODE CM_PTR return_code )
Receive_Mapped_Data - Receiving data and format identifier from a UTM service

A program uses the Receive_Mapped_Data (CMRCVM) call to receive information from a UTM service. The information received can be either data, a format identifier and/or permission to send.

The call can be executed with or without blocking.

- The Receive_Mapped_Data call is “blocking” when the receive_type characteristic has the value CM_RECEIVE_AND_WAIT. If no information (data or permission to send) is present at the time of the Receive_Mapped_Data call, the program run waits in Receive_Mapped_Data until information is available for this conversation. Only then does the program run return from the Receive_Mapped_Data call and bring back the information. If there is information available at the time of the call, the program receives it without waiting.

To limit the wait time for a blocking Receive_Mapped_Data call, appropriate timers should be set in the UTM partner application.

- The Receive_Mapped_Data call is “non-blocking” when the receive_type characteristic has the value CM_RECEIVE_IMMEDIATE. If no information is present at the time of the Receive_Mapped_Data call, the program run does not wait until information for this conversation arrives. The program run returns from the Receive_Mapped_Data call immediately. If there is already information available, it is transferred to the program.

You can set the receive_type characteristic with the Set_Receive_Type call before the Receive_Mapped_Data call.

Syntax

CMRCVM (conversation_ID, map_name, map_name_length, buffer, requested_length, data_received, received_length, status_received, control_information_received, return_code)

Parameters

→ conversation_ID  Identifier of the conversation.

← map_name  Format identifier sent to the CPI-C program by the UTM partner application together with the data. The format identifier specifies the structure information for the received data.

← map_name_length  Length of the format identifier in map_name.
CPI-C calls in UPIC

Receive_Mapped_Data

← buffer
Buffer in which the data is received. If the return value of
\textit{data\_received} is CM\_NO\_DATA\_RECEIVED, the contents of \texttt{buffer}
is undefined.

→ requested_length
Maximum length of data that can be received.

← data\_received
Specifies whether data was received in the conversation.
\textit{data\_received} can have one of the following values:

CM\_NO\_DATA\_RECEIVED
No data was available for the program. Permission to send may
have been received.

CM\_COMPLETE\_DATA\_RECEIVED
A complete message available for the program was received.

CM\_INCOMPLETE\_DATA\_RECEIVED
A message was not transferred in full to the program. If \textit{data\_received}
has this value, the program must issue repeated \texttt{Receive} or
\texttt{Receive\_Mapped\_Data} calls until the message is received in its
entirety, i.e. until \textit{data\_received} has the value
CM\_COMPLETE\_DATA\_RECEIVED.
The value of \textit{data\_received} is undefined if the result of the call is not
CM\_OK or CM\_DEALLOCATED\_NORMAL.

← received\_length
Length of the data received. If the program has not received data
(data\_received=CM\_NO\_DATA\_RECEIVED) or if the result is not
CM\_OK or CM\_DEALLOCATE\_NORMAL, the value of
received\_length is undefined.

← status\_received
Specifies whether the program received permission to send.
\textit{status\_received} can have one of the following values:

CM\_NO\_STATUS\_RECEIVED
Permission to send was not received.

CM\_SEND\_RECEIVED
The UTM service has passed permission to send to the program.
The program must then issue a \texttt{Send\_Data} call.

Unless the return code is CM\_OK, the value of \textit{status\_received} is
undefined.
← control_information_received
This is only supported syntactically and always has the value CM_REQ_TO_SEND_NOT_RECEIVED.

If the return code is not CM_OK or CM_DEALLOCATE_NORMAL, the value of control_information_received is undefined.

← return_code
Result of the function call.

Result (return_code)

CM_OK
The call is OK. The program has one of the following states after function call:

“Receive”, if the value of status_received is CM_NO_STATUS_RECEIVED.
“Send”, if the value of status_received is CM_SEND_RECEIVED.

CM_SECURITY_NOT_VALID
Possible causes:
– an invalid UTM user ID in the Set_Conversation_Security_User_ID call
– an invalid password in the Set_Conversation_Security_Password call
– the UTM application was generated without user IDs (USER statements).
– the user cannot sign on to the UTM application due to a resource bottleneck.

If the UPIC application communicates with an openUTM application that returns a detailed result of the authorization check (Ś V5.1A30), the UPIC library supplies a secondary return code that describes the cause in detail. The results received by the program are listed under secondary_return_code, see page 200.

The secondary return codes can also be queried using the Extract_Secondary_Return_Code call, see page 170.

CM_TPN_NOT_RECOGNIZED
Possible causes:
– a service restart with KDCDISP was rejected as no UTM user ID generated with RESTART=YES was specified.
– an invalid transaction code (TAC) in the upicfile or in the Set_TP_Name call, e.g.:
  – the TAC is not generated
  – you are not authorized to call this TAC
  – the TAC is permitted only as a follow-up TAC
  – the TAC is not a dialog TAC
  – The TAC is generated with encryption but user data was sent without encryption, or encryption is not supported for the connection, or the encrypted data does not have the required encryption level.
– Service restart using KDCDISP was rejected because no UTM user ID generated with RESTART=YES was specified.
CM_TP_NOT_AVAILABLE_NO_RETRY
A service restart with KDCDISP is not possible as the UTM application has been regenerated.

CM_TP_NOT_AVAILABLE_RETRY
A service restart was rejected as the UTM application has been terminated.

CM_DEALLOCATED_ABEND
Possible causes:
- abnormal termination of the UTM service
- termination of the UTM application
- connection shutdown by UTM administration
- connection shutdown by the transport system
- connection shutdown by UTM because the maximum permitted number of users (MAX statement, CONN-USERS=) has been exceeded. This may also occur if an administrator user was transferred in the Set_Conversation_Security_User_ID call but the user ID implicitly assigned to the connection by UTM generation or the (connection) user ID explicitly assigned using the statement LTERM..., USER= is not an administrator user (CONN-USERS applies only for users without administration authorization).

The program enters the “Reset” state.

CM_DEALLOCATED_NORMAL
A PEND-FI call was executed in the UTM service. The program enters the “Reset” state.

CM_OPERATION_INCOMPLETE
The Receive_Mapped_Data call was interrupted by the expiry of the timer that was set with Set_Receive_Timer. No data was received.

CM_UNSUCCESSFUL
The receive_type characteristic has the value CM_RECEIVE_IMMEDIATE and there is currently no data available for the conversation.

CM_Resource_FAILURE_RETRY
A temporary resource bottleneck led to termination of the conversation. It may not be possible to buffer any further data in the UTM page pool.

Remedy: enlarge the UTM page pool (MAX statement, PGPOOL=).

CM_RESOURCE_FAILURE_NO_RETRY
An error occurred which led to premature termination of the conversation (e.g. protocol error or premature loss of network connection).

CM_PROGRAM_STATE_CHECK
The call is not permitted in the current state. The contents of all other variables are undefined.
CM_PROGRAM_PARAMETER_CHECK
The conversation_ID is invalid or the value in requested_length is greater than 32767 or less than 0. The contents of all other variables are undefined.

CM_PRODUCT_SPECIFIC_ERROR
A Receive call was issued instead of a Send_Data call (only directly after an Allocate call).

CM_MAP_ROUTINE_ERROR
In the UTM partner application no format identifiers are supported in the UPIC protocol because an openUTM version below V4.0 is being used on the partner server.

Secondary return code (secondary_return_code)

CM_SECURITY_USER_UNKNOWN
The specified user ID is not generated.

CM_SECURITY_STA_OFF
The specified user ID is locked.

CM_SECURITY_USER_IS_WORKING
Another user is already signed on with this user ID.

CM_SECURITY_OLD_PSWORD_WRONG
The old password entered is incorrect.

CM_SECURITY_NEW_PSWORD_WRONG
The new password information cannot be used. Possible cause: minimum period of validity not yet expired.

CM_SECURITY_NO_CARD_READER
The user is generated with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG
The user is generated with a chipcard and cannot sign on via UPIC.

CM_SECURITY_NO_RESOURCES
Sign-on is not possible at the moment. Possible cause:
– a resource bottleneck, or
– the maximum number of simultaneous users signed on has been reached (see KDCDEF statement MAX CONN-USERS=), or
– an inverse KDCDEF is running.

Try again later.

CM_SECURITY_NO_KERBEROS_SUPPORT
The user is generated with a Kerberos principal and cannot sign on via UPIC.
CM_SECURITY_TAC_KEY_MISSING
The current LTERM is not authorized to resume the service.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
The validity period of the user password has expired.

CM_SECURITY_COMPLEXITY_ERROR
The new password is not sufficiently complex.

CM_SECURITY_PASSWORD_TOO_SHORT
The new password is too short.

CM_SECURITY_UPD_PSWORD_WRONG
The password transferred by KDCUPD does not satisfy the complexity or minimum length requirement defined in application generation.

CM_SECURITY_TA_RECOVERY
A transaction restart is required for the specified user ID.

CM_SECURITY_PROTOCOL_CHANGED
The open service cannot be resumed from this LTERM partner.

CM_SECURITY_SHUT_WARN
The administrator has issued a SHUT WARN. Normal users may no longer sign on to the UTM application, only the administrator may still sign on.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
The encryption mechanism required to resume the open service is not available on the connection.

CM_SECURITY_PWD_EXPIRED_RETRY
The validity period of the user password has expired.

The following secondary return codes only occur in the context of UTM cluster applications:

CM_SECURITY_USER_GLOBALY_UNKNOWN
The specified user ID is not recognized in the cluster user file.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
A user has already signed on to another node application with this user ID.

CM_SECURITY_TRANSIENT_ERROR
A temporary error occurred during signon. The cluster user file could not be accessed in the time configured in the node application.
Try signing on again later.
State change

- If the return code is CM_OK, the program has one of the following states after function call:
  
  "Receive" if the value of status_received is CM_NO_STATUS_RECEIVED.
  "Send" if the value of status_received is CM_SEND_RECEIVED.

- With the following return codes, the program enters the “Reset” state:
  CM_DEALLOCATED_ABEND
  CM_DEALLOCATED_NORMAL
  CM_SECURITY_NOT_VALID
  CM_TPN_NOT_RECOGNIZED
  CM_TPN_NOT_AVAILABLE_RETRY/NO_RETRY
  CM_RESOURCE_FAILURE_RETRY/NO_RETRY
  CM_SECURITY_USER_UNKNOWN
  CM_SECURITY_STA_OFF
  CM_SECURITY_USER_IS_WORKING
  CM_SECURITY_OLD_PSWORD_WRONG
  CM_SECURITY_NEW_PSWORD_WRONG
  CM_SECURITY_NO_CARD_READER
  CM_SECURITY_CARD_INFO_WRONG
  CM_SECURITY_NO_RESOURCES
  CM_SECURITY_NO_KERBEROS_SUPPORT
  CM_SECURITY_TAC_KEY_MISSING
  CM_SECURITY_PWD_EXPIRED_NO_RETRY
  CM_SECURITY_COMPLEXITY_ERROR
  CM_SECURITY_PASSWORD_TOO_SHORT
  CM_SECURITY_UPD_PSWORD_WRONG
  CM_SECURITY_TA_RECOVERY
  CM_SECURITY_PROTOCOL_CHANGED
  CM_SECURITY_SHUT_WARN
  CM_SECURITY_ENC_LEVEL_TOO_HIGH
  CM_SECURITY_PWD_EXPIRED_RETRY
  CM_SECURITY_USER_GLOBALLY_UNKNOWN
  CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
  CM_SECURITY_TRANSIENT_ERROR

- In all other error conditions, the program does not change its state.
Notes

- With a `Receive_Mapped_Data` call, a program can only receive the amount of data specified in the `requested_length` parameter. It is therefore possible that the program has not read the complete message sent by the partner. The `data_received` parameter indicates as shown below whether there is still more message data to be read.
  - If the program has already received the complete message, the `data_received` parameter has the value CM_COMPLETE_DATA_RECEIVED.
  - If the program has not yet received all data of the message, the `data_received` parameter has the value CM_INCOMPLETE_DATA_RECEIVED. The program must then continue to call `Receive_Mapped_Data` or `Receive` until `data_received` has the value CM_COMPLETE_DATA_RECEIVED.

- If a maximum wait time was set with the `Set_Receive_Timer` call before a blocking `Receive_Mapped_Data` call, the program run returns from the `Receive_Mapped_Data` call at the latest once the wait time has expired, and the `Receive_Mapped_Data` call then returns the result (`return_code`) CM_OPERATION_INCOMPLETE.

- A program can use a single call to receive both data and permission to send. The `return_code`, `data_received`, and `status_received` parameters supply details on the kind of information received by a program.

- If the program issues the `Receive_Mapped_Data` call in the “Send” state, permission to send is passed to the UTM service. The send direction of the conversation is thus changed.

- A `Receive` call with `requested_length = 0` has no special meaning. If data is available, it is received in the length 0 with `data_received` = CM_INCOMPLETE_DATA_RECEIVED. If no data is available, permission to send can be received. This means that either data or permission to send can be received, but not both.

- If a message segment is received with `Receive_Mapped_Data` calls (`data_received` has the value CM_INCOMPLETE_DATA RECEIVED except in the last `Receive_Mapped_Data` call), the `map_name` and `map_name_length` parameters are only supplied with values the first time `Receive_Mapped_Data` is called. However, they are not overwritten in the subsequent `Receive_Mapped_Data` calls.

- If the UTM partner application transfers an empty format identifier (i.e. 8 blanks), `map_name` is set to 8 blanks and `map_name_length` to -1.
Behavior in the event of errors

CM_RESOURCE_FAILURE_RETRY
Re-establish conversation. If the error recurs, the page pool of the UTM application may be too small and should be enlarged (MAX statement, PGPOOL=).

CM_RESOURCE_FAILURE_NO_RETRY
Notify the service department and produce a diagnostic report. A fault in the transport system can also cause this return code.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
Modify program.

CM_MAP_ROUTINE_ERROR
Modify program.

CM_OPERATION_INCOMPLETE
The conversation and the communication connection must be explicitly shut down with the Disable_UTM_UPIC call. Any other call can lead to unpredictable results.

CM_SECURITY_USER_UNKNOWN
The UTM user ID is not generated. Use a user ID that is generated or generate or dynamically configure the user ID you want.

CM_SECURITY_STA_OFF
Generate the user ID with STATUS=ON or unlock it using administration facilities.

CM_SECURITY_USER_IS_WORKING
Use another UTM user ID or terminate the service of the user already signed on.

CM_SECURITY_OLD_PSWORD_WRONG
Enter the password correctly.

CM_SECURITY_NEW_PSWORD_WRONG
Use the old password until its validity expires.

CM_SECURITY_NO_CARD_READER
The user is generated with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG
The user is generated with a chipcard.

CM_SECURITY_NO_RESOURCES
Try again later.
CM_SECURITY_NO_KERBEROS_SUPPORT
The user is generated with a Kerberos principal and cannot sign on via UPIC.

CM_SECURITY_TAC_KEY_MISSING
Generation or modify program.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
The validity period of the password has expired. The password must be changed using administration facilities before the user can sign on again.

CM_SECURITY_COMPLEXITY_ERROR
Select a new password that satisfies the requirements of the generated complexity level, see KDCDEF statement USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT
Select a longer password or change generation, see KDCDEF statement USER PROTECT-PW= length, ... (value for the minimum length).

CM_SECURITY_UPD_PSWRD_WRONG
The password is not sufficiently complex or is too short, see KDCDEF statement USER PROTECT-PW=. The password must be changed using administration facilities before the user can sign on again.

CM_SECURITY_TA_RECOVERY
A transaction restart is required for the specified user ID.

CM_SECURITY_PROTOCOL_CHANGED
The user has an open service that cannot be resumed from a UPIC client.

CM_SECURITY_SHUT_WARN
The UTM application is terminated; only users with administration authorization may sign on. Wait until the application has been restarted.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
The encryption mechanism required to resume the open service is not available on the connection.

CM_SECURITY_PWD_EXPIRED_RETRY
Repeat establishment of the conversation using the old password and a new password.
The following secondary return codes only occur in the context of UTM cluster applications:

**CM_SECURITY_USER_GLOBALLY_UNKNOWN**
The specified user ID is not recognized in the cluster user file.

**CM_SECURITY_USER_SIGNED_ON_OTHER_NODE**
A user has already signed on to another node application with this user ID.

**CM_SECURITY_TRANSIENT_ERROR**
A temporary error occurred during signon. The cluster user file could not be accessed in the time configured in the node application.

Try signing on again later.

*Function declaration: Receive_Mapped_Data*

```c
CM_ENTRY Receive_Mapped_Data (unsigned char CM_PTR conversation_ID,
    unsigned char CM_PTR map_name,
    CM_INT32 CM_PTR map_name_length,
    unsigned char CM_PTR buffer,
    CM_INT32 CM_PTR requested_length,
    CM_DATA_RECEIVED_TYPE CM_PTR data_received,
    CM_INT32 CM_PTR received_length,
    CM_STATUS_RECEIVED CM_PTR status_received,
    CM_CONTROL_INFORMATION_RECEIVED CM_PTR request_to_send_received,
    CM_RETURN_CODE CM_PTR return_code )
```
Send_Data - Sending data to a UTM service

A program uses the Send_Data (CMSEND) call to send data to a UTM service. A program must issue a Send_Data or Send_Mapped_Data call each time it receives permission to send. This is the case:

– immediately after a successful Allocate call or
– when status_received has the value CM_SEND_RECEIVED after the Receive or Receive_Mapped_Data call (i.e. when the program has received permission to send).

Syntax

CMSEND (conversation_ID, buffer, send_length, control_information_received, return_code)

Parameters

→ conversation_ID    Identifier of the conversation.
→ buffer             Buffer with the data to be sent. The length of the data is specified in the send_length parameter.
→ send_length        Length in bytes of data to be sent.
                     Minimum: 0, maximum: 32767
                     A Send_Data call with length 0 means that a message with length 0 is sent.
← control_information_received
                     This is only supported syntactically and always has the value CM_REQ_TO_SEND_NOT_RECEIVED.
                     If the return code is not CM_OK, the value of control_information_received is undefined.
← return_code        Result of the function call.
Result (*return_code*)

**CM_OK**
The call is OK.

**CM_TPN_NOT_RECOGNIZED**
This return code can only occur with the first *Send_Data* call after an *Allocate* call. After the conversation was established, an error occurred which led to termination of the conversation.

**CM_DEALLOCATED_ABEND**
Possible causes:
- termination of UTM application
- connection shutdown by UTM administration
- connection shutdown by the transport system

**CM_RESOURCE_FAILURE_RETRY**
A temporary resource bottleneck led to termination of the conversation. It may not be possible to buffer any further data in the UTM page pool.

Action: Increase the size of the UTM page pool (MAX statement PGPOOL=).

**CM_PROGRAM_STATE_CHECK**
The call is not permitted in the current state.

**CM_PROGRAM_PARAMETER_CHECK**
The *conversation_ID* is invalid or the value of *send_length* is greater than 32767 or less than 0.

**State change**

If the return code is CM_OK, the program remains in the “Send” state.

If the return code is CM_TPN_NOT_RECOGNIZED, CM_DEALLOCATED_ABEND, or CM_RESOURCE_FAILURE_RETRY/NO_RETRY, the program enters the “Reset” state.

In all other error conditions, the program does not change its state.

**Note**

UPIC buffers the data to be sent, and does not send it to the UTM server until a later point in time. Consequently, termination of the UTM application may not be returned immediately, and may not be reported until the next call has been issued.
Behavior in the event of errors

CMRESOURCE_FAILURE_RETRY
Re-establish conversation.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

Function declaration: Send_Data

CM_ENTRY Send_Data ( unsigned char CM_PTR conversation_ID,
                     unsigned char CM_PTR buffer,
                     CM_INT32 CM_PTR send_length,
                     CM_CONTROLINFORMATION_RECEIVED CM_PTR control_information_received,
                     CM_RETURN_CODE CM_PTR return_code )
Send_Mapped_Data - Sending data and format identifier

A program uses the Send_Mapped_Data (CMSNDM) call to send data and a format identifier to a UTM service. A program must issue a Send_Data or Send_Mapped_Data call each time it receives permission to send. This is the case

- immediately after a successful Allocate call or
- when status_received has the value CM_SEND_RECEIVED after the Receive or Receive_Mapped_Data call (i.e. when the program has received permission to send).

Syntax

CMSNDM (conversation_ID, map_name, map_name_length, buffer, send_length, control_information_received, return_code)

Parameters

→ conversation_ID  Identifier of the conversation.
→ map_name  Format identifier sent to the UTM application. The format identifier specifies the structure information for the recipient of the data.
→ map_name_length  Length of the format identifier in bytes.
→ buffer  Address of the buffer with the data to be sent. The length of the data is specified in the send_length parameter.
→ send_length  Length in bytes of data to be sent.
  Minimum: 0, maximum: 32767
  A Send_Mapped_Data call with length 0 means that a message with length 0 is sent.
← control_information_received  This is only supported syntactically and always has the value CM_REQ_TO_SEND_NOT_RECEIVED.
  If the return code is not CM_OK, the value of control_information_received is undefined.
← return_code  Result of the function call.
Result *(return_code)*

**CM_OK**
The call is OK.

**CM_TPN_NOT_RECOGNIZED**
This return code can only occur with the first `Send_Mapped_Data` call after an *Allocate* call. After the conversation was established, an error occurred which led to termination of the conversation.

**CM_DEALLOCATED_ABEND**
Possible causes:
- termination of UTM application
- connection shutdown by UTM administration
- connection shutdown by the transport system

**CM_RESOURCE_FAILURE_RETRY**
A temporary resource bottleneck led to termination of the conversation. It may not be possible to buffer any further data in the UTM page pool.

**CM_PROGRAM_STATE_CHECK**
The call is not permitted in the current state.

**CM_PROGRAM_PARAMETER_CHECK**
The *conversation_ID* is invalid or the value of *send_length* is greater than 32767 or less than 0.

**CM_MAP_ROUTINE_ERROR**
Possible causes:
- In the UTM partner application, format identifiers are not supported in the UPIC protocol because an openUTM version lower than V4.0 was used.
- The length of the format identifier is less than 0 or greater than 8.

**State change**

- If the return code is CM_OK, the program remains in the “Send” state.
- If the return code is one of the following the program enters the “Reset” state:
  - CM_TPN_NOT_RECOGNIZED
  - CM_DEALLOCATED_ABEND
  - CM_RESOURCE_FAILURE_RETRY/NO_RETRY
- In all other error conditions, the program does not change its state.
Notes

- The data is always transferred transparently. The data sent is shown to the partner UTM service in the MGET call. The format identifier in `map_name` is transferred to the UTM service in the KCMF/kcfn field during the MGET call.

- For performance reasons, UPIC buffers the data to be sent, and does not send it to the UTM server until later (with a follow-up call). Consequently, termination of the UTM application may not be returned immediately, and may not be reported until the next call has been issued.

- `map_name` is reset as soon as the value of `map_name` is sent to UTM.

Behavior in the event of errors

**CM RESOURCE FAILURE RETRY**
Re-establish conversation. If the error recurs, the page pool of the UTM application may be too small and should be enlarged (MAX statement, PGPOOL=).

**CM PROGRAM STATE CHECK**
Modify program.

**CM PROGRAM PARAMETER CHECK**
Modify program.

Function declaration: Send_Mapped_Data

```c
CM_ENTRY Send_Mapped_Data(unsigned char CM_PTR conversation_ID,
             unsigned char CM_PTR map_name,
             CM_INT32 CM_PTR map_name_length,
             unsigned char CM_PTR buffer,
             CM_INT32 CM_PTR send_length,
             CM_CONTROL_INFORMATION_RECEIVED CM_PTR control_information_received,
             CM_RETURN_CODE CM_PTR return_code )
```
Set_Allocate_Timer - Setting timer for the allocate call

The `Set_Allocate_Timer` call (CMSAT) sets the timeout for an Allocate call.

When this timer is set, the Allocate call is broken off after the time defined in the `allocate_timer` array.

The `Set_Allocate_Timer` call is only permitted in the “Init” state.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C specification.

**UPIC-Local**: Connection via UPIC-Local does not support the `Set_Allocate_Timer` call.

**Syntax**

CMSAT (conversation_ID, allocate_timer, return_code)

**Parameters**

- `conversation_ID`  Conversation identifier
- `allocate_timer`  Time in milliseconds after which an Allocate call is broken off. The Allocate timer is reset if you set `allocate_timer` to 0. The waiting time of the Allocate call is then no longer monitored.
  
  The value specified for `allocate_timer` is rounded up to the next whole second.
- `return_code`  Result of the function call.

**Result (return_code)**

- CM_OK  The call is OK.
- CM_CALL_NOT_SUPPORTED  The function is not supported. This return code only occurs for UPIC-L.
- CM_PROGRAM_STATE_CHECK  The conversation is not in the “Init” state.
- CM_PROGRAM_PARAMETER_CHECK  The value of `conversation_ID` is invalid, or a value < 0 was specified in `allocate_timer`.
- CM_PRODUCT_SPECIFIC_ERROR  The UPIC intance could not be found.
**State change**

If there are no errors the function returns CM_OK. The call does not change the state of the conversation.

**Note**

The Set_Allocate_Timer only makes sense in conjunction with the Allocate call. Set_Allocate_Timer can be called as often as desired between an Initialize_Conversation call and an Allocate call. The value which applies is always the one to have been set when Set_Allocate_Timer was last called prior to an allocate call.

**Behavior in the event of errors**

**CM_CALL_NOT_SUPPORTED**
This is not necessarily an error: If the application is intended for both UPIC-L and UPIC-R this return code just means that the application is linked to a UPIC-L library. If this is the case, timer functions are not possible. The program can take note of this return code and avoid making further calls relating to the timer.

**CM_PROGRAM_STATE_CHECK**
Modify program.

**CM_PROGRAM_PARAMETER_CHECK**
Modify program.

**CM_PRODUCT_SPECIFIC_ERROR**
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

**Function declaration: Set_Allocate_Timer**

```c
CM_ENTRY Set_Allocate_Timer ( unsigned char CM_PTR conversation_ID,
                              CM_TIMEOUT CM_PTR allocate_timer,
                              CM_RETURN_CODE CM_PTR return_code )
```
Set_Client_Context - Setting the client context

The Set_Client_Context (CMSCC) call sets the value for the client context. To simplify restart at the client side, the client can specify and store what is known as a client context openUTM. Whenever the client sends user data to the UTM partner application, the last client context set using the Set_Client_Context function is also sent to the UTM application. The context is buffered by openUTM until the end of the conversation unless it is overwritten with a new context.

If the client requests a restart, the last context saved is transferred back to the client together with the last dialog message.

The client context is not saved by openUTM unless the client is signed on using a UTM user ID with restart functionality. This is a requirement for service restart. The context is ignored in all other cases.

The Set_Client_Context call is permitted only in the "Send" state.

This function is not a component of the CPI-C specification but is an additional function of the UPIC carrier system.

Syntax

CMSCC (conversation_ID, client_context, client_context_length, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ client_context  Specifies the context the client wants to send to openUTM
→ client_context_length  Length of the context
  Minimum 0, maximum: 8
← return_code  Result of the function call
Result (*return_code*)

**CM_OK**
The call is OK

**CM_CALL_NOT_SUPPORTED**
The function is not supported. This return code occurs if no client context can be used because the UTM partner application with Version < 5.0 does not support client context.

**CM_PROGRAM_STATE_CHECK**
The conversation is not in the "Send" state.

**CM_PROGRAM_PARAMETER_CHECK**
The value in `conversation_ID` is invalid or the value of `client_context_length` is less than 0 or more than 8.

**CM_PRODUCT_SPECIFIC_ERROR**
The UPIC instance could not be found.

State change

If there are no errors, the function returns CM_OK. The call does not change the state of the conversation.

Notes

- If the return code is not CM_OK, `client_context` remains unchanged.
- The internal buffer size for the client context is currently limited to 8 bytes.

Behavior in the event of errors

**CM_CALL_NOT_SUPPORTED**
Is not necessarily an error. If a UPIC-R application communicates with various UTM partners, this return code simply means that the application is communicating with a UTM application that cannot receive a client context (openUTM < V5.0). The program can take note of this return code and dispense with further calls relating to client context.

**CM_PROGRAM_STATE_CHECK**
Modify program.

**CM_PROGRAM_PARAMETER_CHECK**
Modify program.
CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Set_Client_Context

CM_ENTRY Set_Client_Context (  
    unsigned char CM_PTR conversation_ID,  
    unsigned char CM_PTR client_context,  
    CM_INT32 CM_PTR client_context_length,  
    CM_RETURN_CODE CM_PTR return_code )
Set_Communication_Protocol - Setting the COMMUNICATION_PROTOCOL conversation characteristic

The Set_Communication_Protocol (CMSCP) call sets the COMMUNICATION_PROTOCOL conversation characteristic.

Set_Communication_Protocol changes the values taken from the side information during the Initialize_Conversation call. The changed values apply only for the duration of a conversation; the values in the side information are not changed.

The Set_Communication_Protocol call cannot be issued after an Allocate call.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

Syntax

CMSCP (conversation_ID, protocol, return_code)

Parameters

→ conversation_ID  Conversation identifier

→ protocol  Specifies which communication protocol is used between the UPIC client and the openUTM server.

The following values can be set for protocol:

CM_COMMUNICATION_PROTOCOL_34
The UPIC client attempts to establish a connection to the openUTM server using protocol 34, e.g. for UTM applications with Version < 4.0.

CM_COMMUNICATION_PROTOCOL_40
The UPIC client attempts to establish a connection to the openUTM server using protocol 40.

← return_code  Result of the function call
Result *(return_code)*

CM_OK
The call is OK

CM_PROGRAM_PARAMETER_CHECK
The value in *conversation_ID* or the protocol version in *protocol* is invalid.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Initialize” state.

State change

The call does not change the state of the conversation.

Note

If the return code is not CM_OK, the characteristic remains unchanged.

Behavior in the event of errors

CM_PROGRAM_STATE_CHECK
Modify program

CM_PROGRAM_PARAMETER_CHECK
Modify program

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Set_Communication_Protocol

CM_ENTRY Set_Communication_Protocol(
    unsigned char                 CM_PTR  conversation_ID,
    CM_COMMUNICATION_PROTOCOL     CM_PTR  protocol,
    CM_RETURN_CODE                CM_PTR  return_code )
Set_Conversation_Encryption_Level - Setting the encryption level

The Set_Conversation_Encryption_Level (CMSCEL) call influences the value of the ENCRYPTION-LEVEL conversation characteristic. The encryption level is used to specify whether during a conversation user data is to be transferred in an encrypted form or not. The call overwrites the value of encryption_level, which was assigned in the Initialize_Conversation call.

The Set_Conversation_Encryption_Level call is only permitted in the “Initialize” state.

UPIC local: The data transfer is protected by the type of transfer being used. The Set_Conversation_Encryption_Level call is not supported.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

Syntax

CMSCEL (conversation_ID, encryption_level, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ encryption_level  Specifies whether the conversation user data is to be transferred in an encrypted or unencrypted form. The following values can be used:

CM_ENC_LEVEL_NONE  The conversation user data is to be transferred in an unencrypted form.

CM_ENC_LEVEL_1  The user data is to be transferred in an encrypted form using the DES algorithm. An RSA key with a key length of 200 bits is used for exchange of the DES key.

CM_ENC_LEVEL_2  The user data is to be transferred in an encrypted form using the AES algorithm. An RSA key with a key length of 512 bits is used for exchange of the AES key.

CM_ENC_LEVEL_3  The user data is to be transferred in an encrypted form using the AES algorithm. An RSA key with a key length of 1024 bits is used for exchange of the AES key.
CM_ENC_LEVEL_4
The user data is to be transferred in an encrypted form using the AES algorithm. An RSA key with a key length of 2048 bits is used for exchange of the AES key.

← return_code Result of the function call.

Result (return_code)

CM_OK
The call is OK.

CM_CALL_NOT_SUPPORTED
The function is not supported. This return code only occurs for UPIC-L. It indicates to the program that encryption is not necessary.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Init” state.

CM_PROGRAM_PARAMETER_CHECK
The value of conversation_ID is invalid, or the value of encryption_level is undefined.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_ENCRYPTION_NOT_SUPPORTED
Encryption is not available for this conversation for one of the following reasons:
– the UTM partner application is based on an openUTM version < V5.0 or the add-on product openUTM-Crypt is not installed.
– the UTM partner application does not want to implement encryption because the UPIC-L client is trusted.
– the UPIC client cannot implement encryption because the product openUTM-Client was installed without the encryption license.

CM_ENCRYPTION_LEVEL_NOT_SUPPORTED
Encryption with the specified encryption level (encryption_level) is not supported by UPIC.

State change
If there are no errors the function returns CM_OK. The call does not change the state of the conversation.
Notes

- If the return code is not CM_OK, the ENCRYPTION_LEVEL characteristic remains unchanged.
- If the encryption level requested by the UTM application is higher than the one on the UPIC client side, the higher encryption level is implemented. Or in other words, if the UTM application requests a certain level of encryption, the UPIC client encrypts the data on this level regardless of the level of encryption set by the UPIC application.
- If there is no communication connection set up to the UTM partner application at the time when the call is made, the function terminates with the CM_OK return code. The system decides when the subsequent Allocate call is made whether the requested encryption level is to be implemented.

Behavior in the event of errors

CM_CALL_NOT_SUPPORTED
Is not necessarily an error: If an application is intended for both UPIC-L and for UPIC-R, this return code just means that the application is linked to a UPIC-L library. In this case encryption is not necessary. The program can take note of this return code and avoid making further calls requesting encryption.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

CM_ENCRYPTION_NOT_SUPPORTED
Is not necessarily an error: If a UPIC-R application is communicating with several UTM partners some of which implement encryption and some of which do not, then this return code just means that it is communicating with an application which either cannot or doesn’t want to implement encryption. In this case encryption is not possible. The program can take note of this return code and avoid making further calls requesting encryption.

CM_ENCRYPTION_LEVEL_NOT_SUPPORTED
The UPIC library has possibly found and loaded an old encryption library (≤ V5.1). Make sure that the encryption library of the latest openUTM client version is installed and is also loaded. Note the search sequence for libraries in the different operating systems.
Function declaration: Set_Conversation_Encryption_Level

CM_ENTRY Set_Conversation_Encryption_Level
    unsigned char CM_PTR conversation_ID,
    CM_ENCRYPTION_LEVEL CM_PTR encryption_level,
    CM_RETURN_CODE CM_PTR return_code )
Set_Conversation_Security_New_Password - Setting new password

The Set_Conversation_Security_New_Password (CMSCSN) call sets the value for the conversation characteristics security_new_password and security_new_password_length. The security_new_password is understood as the new password of a UTM user ID.

A program can only specify a new password if the security_type characteristic is set to CM_SECURITY_PROGRAM.

The call cannot be issued after an Allocate call.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

Syntax

CMSCSN (conversation_ID, security_new_password, security_new_password_length, return_code)

Parameters

→ conversation_ID Identifier of the conversation.
→ security_new_password Password which is to replace the old password. The UTM partner application uses this new password to replace the old password following a valid access authorization with the old password.
→ security_new_password_length Length in bytes of the password specified in security_new_password. Minimum: 0, maximum: 8
  If you specify 0 here, security_new_password is filled with 8 blanks, i.e. UTM does not alter the existing password.
← return_code Result of the function call.
Result *(return_code)*

**CM_OK**
The call is OK.

**CM_CALL_NOT_SUPPORTED**
The function is not supported. No new password is possible for this conversation because the UTM partner application is based on an openUTM version < V5.1.

**CM_PROGRAM_STATE_CHECK**
The conversation is not in the “Initialize” state or *security_type* is not set to CM_SECURITY_PROGRAM.

**CM_PROGRAM_PARAMETER_CHECK**
The value of *conversation_ID* is invalid, the value in *security_new_password_length* is less than 0 or greater than 8, or the new password only comprises blanks.

**CM_PRODUCT_SPECIFIC_ERROR**
The UPIC instance could not be found.

If the return code is not CM_OK, the *security_new_password* and *security_new_password_length* characteristics remain unchanged.

**State change**
The call does not change the state of the conversation.

**Notes**
- If a program calls *Set_Conversation_Security_New_Password*, a user ID must also be specified. The user ID is set in the program using the *Set_Conversation_Security_User_ID* call.
- An invalid password is not detected with this call. The partner application checks the password for validity after the conversation is established. If the password is invalid, the partner application issues an error message which is stored in the UPIC log file.
- The program is notified of the incorrect password by means of the return code CM_SECURITY_NOT_VALID. This is returned following a CPI-C call issued after the *Allocate* call.
● If several conversations are established with the same partner application (or in other words, the communication connection is not set up and cleared down every time), the result of CMSCSN will be CMINIT CM_OK after the first call, but after all subsequent CMINIT calls it will be CM_CALL_NOT_SUPPORTED. The UPIC library only establishes a connection to the partner application after the first CMALLC call and thus can only then determine whether the version of the partner application supports password changes.

Following the first CMSCSN call with the result CM_OK, the program only detects that there is no support for password changes by means of the return code CM_SECURITY_NOT_SUPPORTED.

This is returned following an Allocate call.

● If only blanks were specified for the new password, this means the UTM application should reset the password, i.e. the user no longer requires a password. However, this is not permitted from the client, so consequently the error CM_PROGRAM_PARAMETER_CHECK is returned.

**Behavior in the event of errors**

CM_PROGRAM_PARAMETER_CHECK

Modify program.

CM_PRODUCT_SPECIFIC_ERROR

The operating system cannot provide sufficient memory for the internal buffers.

Check whether the memory requirements of your program are too high and if necessary reboot your system.

**Function declaration: Set_Conversation_Security_New_Password**

```c
CM_ENTRY Set_Conversation_Security_New_Password (
    unsigned char CM_PTR conversation_ID,
    unsigned char CM_PTR security_new_password,
    CM_INT32 CM_PTR security_new_password_length,
    CM_RETURN_CODE CM_PTR return_code )
```
**Set_Conversation_Security_Password - Setting the password**

The `Set_Conversation_Security_Password` (CMSCSP) call sets the values for the conversation characteristics `security_password` and `security_password_length`. The `security_password` is understood as the password of a UTM user ID.

A program can only specify a password if the `security_type` characteristic is set to CM_SECURITY_PROGRAM.

The call cannot be issued after an Allocate call.

This function is one of the advanced functions.

**Syntax**

`CMSCSP (conversation_ID, security_password, security_password_length, return_code)`

**Parameters**

→ `conversation_ID` Identifier of the conversation.

→ `security_password`  
Password used to establish the conversation. The UTM partner application uses this password together with the user ID in order to check access authorization.

The password is specified in the local code used on the machine and converted into EBCDIC if necessary (see section “Code conversion” on page 104).

→ `security_password_length`  
Length in bytes of the password specified in `security_password`.  
Minimum: 0, maximum: 8  
If you specify 0 here, `security_password` is filled with 8 blanks, i.e. no password is transferred to openUTM for checking access authorization.

← `return_code` Result of the function call.
Set_Conversation_Security_Password

Result (return_code)

CM_OK
  The call is OK.

CM_PROGRAM_STATE_CHECK
  The conversation is not in the “Initialize” state or security_type is not set to
  CM_SECURITY_PROGRAM.

CM_PROGRAM_PARAMETER_CHECK
  The conversation_ID is invalid or the value in security_password_length is less than 0
  or greater than 8.

CM_PRODUCT_SPECIFIC_ERROR
  The UPIC instance could not be found.

If the return code is not CM_OK, the security_password and security_password_length character-
istics remain unchanged.

State change

None.

Notes

● If a program calls Set_Conversation_Security_Password, a user ID must also be specified. The user ID is set in the program using the Set_Conversation_Security_User_ID call.

● An invalid password is not detected with this call. The partner application checks the password for validity after the conversation is established. If the password is invalid, the partner application issues an error message which is stored in the UPIC log file (see section “UPIC log file” on page 363).

● The program is notified of the incorrect password by means of the return code CM_SECURITY_NOT_VALID. This is returned following a CPI-C call issued after the Allocate call.
Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

Function declaration: Set_Conversation_Security_Password

CM_ENTRY Set_Conversation_Security_Password (unsigned char CM_PTR conversation_ID,
unsigned char CM_PTR security_password,
CM_INT32 CM_PTR security_password_length,
CM_RETURN_CODE CM_PTR return_code)
Set_Conversation_Security_Type - Setting the security type

The `Set_Conversation_Security_Type` (CMSCST) call sets the value for the conversation characteristic `security_type`.

The call overwrites the value assigned in the `Initialize_Conversation` call, and must not be executed after the `Allocate` call.

This function is one of the advanced functions.

Syntax

```c
CMSCST (conversation_ID, security_type, return_code)
```

Parameters

- `conversation_ID` Identifier of the conversation.
- `security_type` Specifies the type of access information sent when establishing the conversation with the partner application. This information is used by the partner application to check access authorization.

  *The following values can be set for security_type:*

  - `CM_SECURITY_NONE` No access information is transferred to the partner application.
  - `CM_SECURITY_PROGRAM` The values of the `security_user_ID` and `security_password` characteristics are used as access information. This means that the access information consists of:
    - either a UTM user ID
    - or a UTM user ID and a password.

- `return_code` Result of the function call.

  *Result (return_code)*

  - `CM_OK` The call is OK.
  - `CM_PROGRAM_STATE_CHECK` The conversation is not in the “Initialize” state.
CM_PROGRAM_PARAMETER_CHECK
The conversation_ID is invalid or the value in security_type is undefined.

CM_PARM_VALUE_NOT_SUPPORTED
A value not supported by CPI-C has been entered in security_type.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

If the return code is not CM_OK, the security_type characteristic remains unchanged.

State change
None.

Notes
- If the value CM_SECURITY_PROGRAM is entered in security_type, the user ID and possibly the password must be set using the following calls: Set_Conversation_Security_User_ID and Set_Conversation_Security_Password.
- If only the user ID is required for the access check, the Set_Conversation_Security_Password call is not necessary.

Behavior in the event of errors
CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PARM_VALUE_NOT_SUPPORTED
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high, and if necessary reboot your system.

Function declaration: Set_Conversation_Security_Type

CM_ENTRY Set_Conversation_Security_Type (  
unsigned char CM_PTR conversation_ID,  
CM_CONVERSATION_SECURITY_TYPE CM_PTR conversation_security_type,  
CM_RETURN_CODE CM_PTR return_code )
Set_Conversation_Security_User_ID - Setting the UTM user ID

The Set_Conversation_Security_User_ID (CMSCSU) call sets the values for the conversation characteristics security_user_ID and security_user_ID_length. The security_user_ID is understood as a user ID of a UTM application.

A program can only specify a user ID if the security_type characteristic is set to CM_SECURITY_PROGRAM.

The call must not be executed after the Allocate call.

This function is one of the advanced functions.

Syntax

CMSCSU (conversation_ID, security_user_ID, security_user_ID_length, return_code)

Parameters

→ conversation_ID Identifier of the conversation.
→ security_user_ID User ID used to establish the conversation. The UTM partner application uses the user ID and possibly the password to check access authorization.

The partner application may also use the user ID for logging or accounting purposes.

→ security_user_ID_length Length in bytes of the user ID specified in security_user_ID.

Minimum: 0, maximum: 8

If 0 is specified here, despite the fact that security_type is set to CM_SECURITY_PROGRAM in the Set_Conversation_Security_Type call, a connection is not set up to UTM (error in the Allocate call).

← return_code Result of the function call.

Result (return_code)

CM_OK The call is OK.

CM_PROGRAM_STATE_CHECK The conversation is not in the “Initialize” state or security_type is not set to CM_SECURITY_PROGRAM.
CM_PROGRAM_PARAMETER_CHECK
The conversation_ID is invalid or the value in security_user_ID_length is less than 0 or greater than 8.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

If the return code is not CM_OK, the security_user_ID and security_user_ID_length characteristics remain unchanged.

State change
None.

Notes
- The call does not check the user ID for validity. This is carried out by the partner application after the conversation is established. If the user ID is invalid, the UTM server rejects the conversation
- The program is notified of an invalid user ID or an incorrect password by means of the return code CM_SECURITY_NOT_VALID. This is returned following a Receive call issued after the Allocate call.
- If the security_type parameter is set to CM_SECURITY_NONE in the Set_Conversation_Security_Type call, the Set_Conversation_Security_User_ID call is not permitted.

Behavior in the event of errors
CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

Function declaration: Set_Conversation_Security_User_ID

CM_ENTRY Set_Conversation_Security_User_ID (  
unsigned char CM_PTR conversation_ID,  
unsigned char CM_PTR security_user_ID,  
CM_INT32 CM_PTR security_user_ID_length,  
CM_RETURN_CODE CM_PTR return_code )
Set_Conversion - Setting the CHARACTER_CONVERSION conversation characteristic

The Set_Conversion (CMSCNV) call sets the CHARACTER_CONVERSION conversation characteristic.

Set_Conversion changes the values that were taken from the side information during the Initialize_Conversation call. The changed values apply only for the duration of a conversation. The values in the side information are not changed.

The Set_Conversion call can no longer be issued after the Allocate call.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

Syntax

CMSCNV (conversation_ID, character_conversion, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ character_conversion  Specifies whether code conversion for the user data is to be performed or not.

The following values can be set for character_conversion:

CM_NO_CHARACTER_CONVERSION  There is no automatic code conversion when data is sent or received.

CM_IMPLICIT_CHARACTER_CONVERSION  Data is automatically converted when sent or received (see also section “Code conversion” on page 104).

← return_code  Result of the function call
Result *(return_code)*

CM_OK
  The call is OK

CM_PROGRAM_PARAMETER_CHECK
  The value in *conversation_ID* or the value for *CHARACTER_CONVERTION* is invalid.

CM_PRODUCT_SPECIFIC_ERROR
  The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK
  The conversation is not in the “Initialize” state.

State change

The call does not change the state of the conversation.

Note

If the return code is not CM_OK, the characteristic remains unchanged.

Behavior in the event of errors

CM_PROGRAM_STATE_CHECK
  Modify program.

CM_PROGRAM_PARAMETER_CHECK
  Modify program.

CM_PRODUCT_SPECIFIC_ERROR
  The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: *Set_Conversion*

```
CM_ENTRY Set_Conversion(
    unsigned char                  CM_PTR  conversation_ID,
    CM_CHARACTER_CONVERTION_TYPE   CM_PTR  conversion_type,
    CM_RETURN_CODE                 CM_PTR  return_code )
```
Set_Deallocate_Type - Setting deallocate_type

A program uses the `Set_Deallocate_Type` (CMSDT) call to set the value of the conversation characteristic `deallocate_type`.

This call is one of the advanced functions.

**Syntax**

CMSDT (conversation_ID, deallocate_type, return_code)

**Parameters**

→ conversation_ID  Identifier of the conversation.
→ deallocate_type  Specifies the type of deallocation for a conversation.

*deallocate_type* must have the value CM_DEALLOCATE_ABEND.

← return_code  Result of the function call.

**Result (return_code)**

CM_OK

The call is OK.

CM_PROGRAM_PARAMETER_CHECK

The `conversation_ID` is invalid or the value of `deallocate_type` is out of range. The value of `deallocate_type` remains unchanged.

CM_PRODUCT_SPECIFIC_ERROR

The value of `deallocate_type` is not CM_DEALLOCATE_ABEND. The value of `deallocate_type` remains unchanged.

**State change**

None.

**Note**

The `deallocate_type` CM_DEALLOCATE_ABEND is used by a program to terminate a conversation unconditionally (regardless of the current state). This type of deallocation should be carried out by the program only in exceptional circumstances.
Behavior in the event of errors

CM_PROGRAM_SPECIFIC_ERROR
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

Function declaration: Set_Deallocate_Type

CM_ENTRY Set_Deallocate_Type ( unsigned char CM_PTR conversation_ID, CM_DEALLOCATE_TYPE CM_PTR deallocate_type, CM_RETURN_CODE CM_PTR return_code )
Set_Function_Key - Setting a UTM function key

The **Set_Function_Key** (CMSFK) call sets the value for the `function_key` characteristic. `function_key` specifies a function key of the UTM partner application.

The value of `function_key` is transferred to the UTM application together with the data of the next `Send_Data` or `Send_Mapped_Data` call, and the function assigned to this function key in the UTM application is executed. The CPI-C program has in effect “pressed the function key”.

The **Set_Function_Key** call is only permitted in the “Send” or “Receive” states.

**Set_Function_Key** is not part of the CPI-C Specification, but is an additional function of the UPIC carrier system.

**Syntax**

CMSFK (conversation_ID, function_key, return_code)

**Parameters**

- **→ conversation_ID**  
  Identifier of the conversation

- **→ function_key**  
  “Function key” that the local CPI-C program wants to “press” in the remote UTM application.

  The function keys must be specified in the format CM_FKEY_fkey, where `fkey` is the number of the K or F key to be “pressed”.  
  Example: if function key F10 of the UTM partner application is to be “pressed”, you must specify for `function_key` the value CM_FKEY_F10.

  openUTM on Unix systems and Windows systems supports the function keys F1 through F20. openUTM on BS2000/OSD supports the function keys K1 through K14 and F1 through F24.

  The value CM_UNMARKED means that no function key is set.

- **← return_code**  
  Result of the function call.
Result *(return_code)*

CM_OK
The call is OK.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Send” or “Receive” state.

CM_PROGRAM_PARAMETER_CHECK
The *conversation_ID* or *function_key* is invalid.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_MAP_ROUTINE_ERROR
In the UTM partner application, function keys are not supported in the UPIC protocol because an openUTM version lower than V4.0 is being used on the partner server.

State change

If there are no errors, this function returns the result CM_OK. This call does not change the state of the program.

Notes

- With openUTM on Unix systems and Windows systems, function keys are only effective in format mode, i.e. when the *Send_Mapped_Data* and *Receive_Mapped_Data* calls are used to exchange data.

- The function key specified in *Set_Function_Key* is only transferred to the UTM partner application together with the data of the subsequent *Send_Data* or *Send_Mapped_Data* call.

  As soon as the value of *function_key* is sent to UTM, *function_key* is reset to CM_UNMARKED (no function key) in the local CPI-C program.

- If the UTM partner application receives a function key from a UPIC client, only the RET parameter of the SFUNC control statement which describes the function key is interpreted. RET contains the return code which appears in the KCRCCC field of the communication area after the MGET call of the UTM service. If the RET parameter is not generated for the function key, UTM always supplies the return code 19Z with the MGET call (function key not generated or special function invalid).
**Behavior in the event of errors**

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

**Function declaration: Set_FUNCTION_KEY**

CM_ENTRY Set_FUNCTION_KEY ( unsigned char CM_PTR conversation_ID,
                                 CM_INT32 CM_PTR function_key,
                                 CM_RETURN_CODE CM_PTR return_code)
Set_Partner_Host_Name - Setting the partner host name

The *Set_Partner_Host_Name* (CMSPHN) call sets the value for the *HOSTNAME* characteristic of the partner application of the conversation. The call overwrites the value which was assigned using the *Initialize_Conversation* call. After an *Allocate* call it may no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

**UPIC-Local:**

The *Set_Partner_Host_Name* call is not supported for connection over UPIC-L.

**UPIC-R using openUTM clusters:**

The *Set_Partner_Host_Name* call is not supported if an openUTM cluster is configured.

**Syntax**

CMSPHN (conversation_ID, host_name, host_name_length, return_code)

**Parameters**

→ conversation_ID  Conversation identifier

→ host_name  Specifies which host name is to be used.

→ host_name_length  Specifies the length of *host_name* in bytes. Minimum:1, maximum:32

← return_code  Result of the function call

**Result (return_code)**

**CM_OK**

The call is OK

**CM_CALL_NOT_SUPPORTED**

This return code always occurs in UPIC-L. It indicates to the program that a *host_name* cannot be used because UPIC-L does not need this information as a result of the underlying communication system.

The return code only occurs with UPIC-R if an openUTM cluster has been configured. It indicates to the program that the *host_name* cannot be modified.
Set_Partner_Host_Name

CPI-C calls in UPIC

CM_PROGRAM_PARAMETER_CHECK
The value of conversation_ID or host_name_length is invalid.

CM_PROGRAM_STATE_CHECK
The conversation is in the “Init” state.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

State change
The call does not change the state of the conversation.

Note
The value of host_name is ignored if there is also a value set for ip_address, either in the upicfile or using a Set_Partner_IP_Address call in the UPIC program.

Behavior in the event of errors
CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

CM_CALL_NOT_SUPPORTED
This is not necessarily an error: The program can take note of this return code and avoid making further calls to set address information.

Function declaration: Set_Partner_Host_Name

CM_ENTRY Set_Partner_Host_Name( unsigned char CM_PTR conversation_ID, unsigned char CM_PTR host_name, CM_INT32 CM_PTR host_name_lth, CM_RETURN_CODE CM_PTR return_code )
Set_Partner_IP_Address - Setting the IP address of the partner application

The Set_Partner_IP_Address (CMSPIA) call sets the value for the IP-ADDRESS characteristic of the conversation. The call overwrites the value assigned using Initialize_Conversation call. After the Allocate call, this call can no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

UPIC-Local:
The Set_Partner_IP_Address call is not supported for connection over UPIC-L.

UPIC-R using openUTM clusters:
The Set_Partner_IP_Address call is not supported if an openUTM cluster is configured.

Syntax
CMSPIA (conversation_ID, ip_address, ip_address_length, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ ip_address  Specifies that an IP address is to be used instead of a hostname characteristic.
→ ip_address_length  Specifies the length of ip_address in bytes. Minimum:0, maximum:64.
← return_code  Result of the function call.
Set_Partner_IP_Address

CPI-C calls in UPIC

Result (*return_code*)

CM_OK
The call is OK.

CM_CALL_NOT_SUPPORTED
The function is not supported.

X/W
This code is always returned with UPIC-L. It indicates to the program that an
ip_address cannot be used because UPIC-L does not need this information as a
result of the underlying communication systems.

The return code only occurs with UPIC-R if an openUTM cluster has been
configured. It indicates to the program that the ip_address cannot be modified.

B
The code is returned with UPIC-R for BS2000/OSD in the event that the UPIC
library on BS2000 is used together with CMX. The CMX communication system
used by UPIC-R does not provide any option on BS2000 systems for passing IP
addresses for addressing the partner application at the interface.

CM_PROGRAM_PARAMETER_CHECK
The value of conversation_ID or ip_address_length is invalid.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Init” state.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

State change
The call does not change the state of the conversation.

Notes

- For IPv4, ip_address is specified using the usual dot notation:

  xxx.xxx.xxx.xxx

  The individual octets xxx are restricted to 3 digits. The contents of the octet are always
interpreted as a decimal number. In particular, this means that octets which are padded
with leading zeros not interpreted as octal numbers.
CPI-C calls in UPIC

- **ip_address** is specified for IPv6 using normal colon notation:
  \[ x:x:x:x:x:x:x:x \]
  
  'x' is a hexadecimal number between 0 and FFFF. The alternative methods of writing IPv6 addresses are permitted (see RFC2373).

  If an embedded IPv4 address in dot notation is specified in the IPv6 address, the above also supplies to the octet for the IPv4 address. The octets are always interpreted as octal numbers.

- If both **ip_address** and HOST_NAME are set, the value of HOST_NAME is ignored.

### Behavior in the event of errors

**CM_PROGRAM_PARAMETER_CHECK**  
Modify program.

**CM_PROGRAM_STATE_CHECK**  
Modify program.

**CM_PRODUCT_SPECIFIC_ERROR**  
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

**CM_CALL_NOT_SUPPORTED**  
This is not necessarily an error: The program can take note of this return code and avoid making further calls to set address information.

On BS2000/OSD, this return code means that the application is connected to UPIC-R and CMX. The program can remember this return code and then no longer requires the **Set_Partner_IP_Address** and **Set_Partner_Port** calls.

### Function declaration: Set_Partner_IP_Address

```
CM_Entry Set_Partner_IP_Address ( unsigned char CM_PTR conversation_ID,  
                               unsigned char CM_PTR ip_address,  
                               CM_INT32 CM_PTR ip_address_length,  
                               CM_RETURN_CODE CM_PTR return_code )
```
Set_Partner_LU_Name - Setting the conversation characteristics partner_LU_name

The Set_Partner_LU_Name call (CMSPLN) sets the conversation characteristics partner_LU_name and partner_LU_name_length.

Set_Partner_LU_Name changes the values taken from the side information in the Initialize_Conversation call. The changed values only apply for the duration of a conversation; the values in the side information itself are not changed.

The Set_Partner_LU_Name call cannot be executed after the Allocate call.

This call is one of the advanced functions.

UPIC-R using openUTM clusters:

The Set_Partner_LU_Name call is not supported if an openUTM cluster is configured.

Syntax

CMSPLN (conversation_ID, partner_LU_name, partner_LU_name_length, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ partner_LU_name  Defines which partner_LU_name should be used.
→ partner_LU_name_length  Specifies the length of partner_LU_name.
  Minimum: 1, maximum: 32.
  UPIC-L:
    Minimum: 1, maximum: 8.
← return_code  Result of the function call.

Result (return_code)

CM_OK  The call is OK.

CM_PROGRAM_PARAMETER_CHECK
  The value of conversation_ID is invalid or partner_LU_name is invalid or the value in partner_LU_name_length is less than 1 or greater than 32.

CM_PROGRAM_STATE_CHECK
  The conversation is not in "Initialize" state.
CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_CALL_NOT_SUPPORTED
This function is not supported.

The return code occurs with UPIC-R if an openUTM cluster has been configured. It indicates to the program that the partner_LU_name cannot be modified.

State change
The call does not change the state of the conversation.

Notes

- If the return code is not CM_OK, the partner_LU_name characteristic remains unchanged.

- This call only sets the partner_LU_name characteristic. An invalid partner_LU_name is not detected with this call. Only the Allocate call detects an invalid partner_LU_name, if it is unable to establish a transport connection to the UTM application. In this case, it returns the CM_ALLOCATE_FAILURE_NO_RETRY return code.

- The Set_Partner_LU_Name call returns CM_OK if an application is linked with UPIC-L and passes a partner_LU_name with a length > 8. However, the partner_LU_name is cut off after 8 bytes without notification in the following Allocate call.

Behavior in the event of errors

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

CM_CALL_NOT_SUPPORTED
Is not necessarily an error: The program can remember this return code and no longer issue any calls for setting address information.
Function declaration: Set_Partner_LU_Name

CM_ENTRY Set_Partner_LU_Name ( unsigned char CM_PTR conversation_ID,
                            unsigned char CM_PTR partner_LU_name,
                            CM_INT32 CM_PTR partne_LU_name_length,
                            CM_RETURN_CODE CM_PTR return_code )
Set_Partner_Port - Setting the TCP/IP port for the partner application

The *Set_Partner_Port* (CMSPP) call sets the port number for TCP/IP for the partner application and in doing so also sets the *PORT* conversation characteristic. The call overwrites the value assigned using the *Initialize_Conversation* call. It may no longer be issued after an *Allocate* call.

The function is one of the additional functions of the UPIC carrier systems; it is not a component of the CPI-C interface.

**UPIC-Local:**

Connection via UPIC local does not support the Set_Partner_Port call.

**Syntax**

CMSPP (conversation_ID, listener_port, return_code)

**Parameters**

→ conversation_ID  Conversation identifier
→ port_number  Specifies which port number is searched for in the communication system by the partner application.

Minimum: 0; maximum: 32767

← return_code  Result of the function call

**Result (return_code)**

CM_OK  The call is OK.

CM_CALL_NOT_SUPPORTED  The function is not supported. This return code occurs for UPIC-L and for UPIC-R on BS2000/OSD.

This code is always returned with UPIC-L. It indicates to the program that a port number cannot be assigned because UPIC-L does not require this information as a result of the underlying communication system.

The code is only returned with UPIC-R (BS2000) in the event that the UPIC library on BS2000 is used together with CMX. The CMX communication system used by UPIC-R does not provide any option on BS2000 systems for passing IP addresses for addressing the partner application at the interface. If the UPIC library uses the Socket interface as its communication system, the code is never returned.
Set_Partner_Port

CPI-C calls in UPIC

CM_PROGRAM_PARAMETER_CHECK
The value of conversation_ID or port_number is invalid.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Init” state.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

State change
The call does not change the state of the conversation.

Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers.
Check whether the memory requirements of your program are too high and if necessary reboot your system.

CM_CALL_NOT_SUPPORTED
This is not necessarily an error: If the application is intended for both UPIC-L and UPIC-R on Unix systems and Windows systems this return code just means that the application is linked to a UPIC-L library. The program can take note of this return code and avoid making further calls to set address information.

On BS2000/OSD, this return code means that the application is connected to UPIC-R and CMX. The program can remember this return code and then no longer requires the Set_Partner_IP_Address and Set_Partner_Port calls.

Function declaration: Set_Partner_Port

CM_ENTRY Set_Partner_Port ( unsigned char CM_PTR conversation_ID,
CM_INT32 CM_PTR port_number,
CM_RETURN_CODE CM_PTR return_code )
Set_Partner_Tsel - Setting the T-SEL of the partner application

The Set_Partner_Tsel (CMSPT) call sets the value for the T-SEL characteristic of the partner application of the conversation. The call overwrites the value assigned using the Initialize_Conversation call. After the Allocate call, this call may no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

UPIC-Local:
Connection via UPIC local does not support the Set_Partner_Tsel call.

Syntax

CMSPT (conversation_ID, transport_selector, transport_selector_length, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ transport_selector  Transport selector of the partner application which is transferred to the communication system.
→ transport_selector_length  Length of the transport selector in bytes.
  Minimum: 0, maximum: 8
  If the length of the transport selector is entered as 0, the first name part of the partner_LU_name is used as the transport selector.
← return_code  Result of the function call.
Result (*return_code*)

CM_OK
The call is OK.

CM_CALL_NOT_SUPPORTED
The function is not supported. This return code only occurs for UPIC-L. It indicates to the program that a TSEL cannot be allocated because UPIC-L does not need this information as a result of the underlying communication system.

CM_PROGRAM_PARAMETER_CHECK
   The value of either *conversation_ID* or *transport_selector_length* is invalid.

CM_PROGRAM_STATE_CHECK
   The conversation is not in the “Init” state.

CM_PRODUCT_SPECIFIC_ERROR
   The UPIC instance could not be found.

State change
The call does not change the state of the conversation.

Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
   Modify program.

CM_PROGRAM_STATE_CHECK
   Modify program.

CM_PRODUCT_SPECIFIC_ERROR
   The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

CM_CALL_NOT_SUPPORTED
   This is not necessarily an error: If the application is intended for both UPIC-L and UPIC-R this return code just means that the application is linked to a UPIC-L library. The program can take note of this return code and avoid making further calls to set address information.

Function declaration: *Set_Partner_Tsel*

```
CM_ENTRY Set_Partner_TSEL ( unsigned char CM_PTR conversation_ID,
                            unsigned char CM_PTR transport_selector,
                            CM_INT32 CM_PTR transport_selector_length,
                            CM_RETURN_CODE CM_PTR return_code )
```

Set_Partner_Tsel_Format - Setting the T-SEL format of the partner application

The Set_Partner_Tsel_Format (CMSPTF) call sets the value for the T-SEL-FORMAT characteristic of the partner application of the conversation. The call overwrites the value assigned using the Initialize_Conversation call. After the Allocate call, this call can no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

UPIC-Local:
Connection via UPIC local does not support the Set_Partner_Tsel_Format call.

Syntax

CMSPTF (conversation_ID, tsel_format, return_code)

Parameters

→ conversation_ID  Conversation identifier
→ tsel_format  Specifies which character set is to be used for the transport selector (TSEL). The following values can be entered:
  – CM_TRANSDATA_FORMAT
    The transport selector is transferred to the communication system using TRANSDATA format.
  – CM_EBCDIC_FORMAT
    The transport selector is transferred to the communication system using EBCDIC format.
  – CM_ASCII_FORMAT
    The transport selector is transferred to the communication system using ASCII format.
← return_code  Result of the function call.
Result (*return_code*)

CM_OK
   The call is OK.

CM_CALL_NOT_SUPPORTED
   The function is not supported. This return code only occurs in UPIC-L. It indicates
to the program that a TSEL format cannot be assigned because UPIC-L does not
require this information as a result of the underlying communication system.

CM_PROGRAM_PARAMETER_CHECK
   The value of either *conversation_ID* or *tsel_format* is invalid.

CM_PROGRAM_STATE_CHECK
   The conversation is not in the “Init” state.

CM_PRODUCT_SPECIFIC_ERROR
   The UPIC instance could not be found.

State change

The call does not change the state of the conversation.

Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
   Modify program.

CM_PROGRAM_STATE_CHECK
   Modify program.

CM_PRODUCT_SPECIFIC_ERROR
   The operating system cannot provide sufficient memory for the internal buffers.
   Check whether the memory requirements of your program are too high and if
   necessary reboot your system.

CM_CALL_NOT_SUPPORTED
   This is not necessarily an error: If the application is intended for both UPIC-L and
   UPIC-R this return code just means that the application is linked to a UPIC-L library.
   The program can take note of this return code and avoid making further calls to set
   address information.

Function declaration: *Set_Partner_TSEL_Format*

```c
CM_ENTRY Set_Partner_TSEL_Format ( unsigned char CM_PTR conversation_ID,
                                  CM_TSEL_Format CM_PTR tsel_format, 
                                  CM_RETURN_CODE CM_PTR return_code )
```
Set.Receive_Timer - Setting the timer for a blocking receive

The Set.Receive_Timer (CMSRCT) call sets the timeout timer for a blocking Receive or Receive_Mapped_Data call.

When this timer is set and receive_type=CM_RECEIVE_AND_WAIT is set for receiving data, the Receive and Receive_Mapped_Data calls are aborted after the period of time defined in the receive_timer field.

Set.Receive_Timer can be called after the Allocate call at any time and as often as you like within a conversation. The timer setting of the last Set.Receive_Timer call applies in each case.

This function is not part of the CPI-C Specification, but is an additional function of the UPIC carrier system.

UPIC local:
Connection via UPIC local does not support the Set.Receive_Timer call.

Syntax
CMSRCT (conversation_ID, receive_timer, return_code)

Parameters

→ conversation_ID Identifier of the conversation

→ receive_timer Time in milliseconds after which a blocking Receive or Receive_Mapped_Data call is interrupted. The Receive and Receive_Mapped_Data calls have a blocking effect when the receive_type characteristic has the value CM_RECEIVE_AND_WAIT. The receive timer is reset when you set receive_timer to 0. The wait time of the Receive or Receive_Mapped_Data call is then no longer monitored.

The value specified for receive_timer is rounded up to the next full second.

← return_code Result of the function call.
Result *(return_code)*

CM_OK
The call is OK.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Send” or “Receive” state.

CM_PROGRAM_PARAMETER_CHECK
*conversation_ID* is invalid or a value < 0 was specified in *receive_timer*.

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_CALL_NOT_SUPPORTED
The function is not supported.

State change

If there are no errors, this function returns the result CM_OK. This call does not change the state of the conversation.

Notes

- The *Set_Receive_Timer* is only useful in connection with the *Receive* and *Receive_Mapped_Data* calls.

- *Set_Receive_Timer* can be called an unlimited number of times within a conversation. The valid value is always the one which was set in the last call of *Set_Receive_Timer* before a *Receive* or *Receive_Mapped_Data* call. The value set remains valid until the next *Set_Receive_Timer* call or until the end of the conversation.
**Behavior in the event of errors**

**CM_PROGRAM_STATE_CHECK**
Modify program.

**CM_PROGRAM_PARAMETER_CHECK**
Modify program.

**CM_PRODUCT_SPECIFIC_ERROR**
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

**CM_CALL_NOT_SUPPORTED**
This is not necessarily an error: If the application is intended for both UPIC-L and UPIC-R this return code just means that the application is linked to a UPIC-L library. The program can take note of this return code and avoid making further Set_Receive_Timer calls.

**Function declaration: Set_Receive_Timer**

```c
CM_ENTRY Set_Receive_Timer ( unsigned char CM_PTR conversation_ID,
                             CM_TIMEOUT CM_PTR timeout_time,
                             CM_RETURN_CODE CM_PTR return_code )
```
Set_Receive_Type - Setting the receive type

The Set_Receive_Type (CMSRT) call sets the value for the conversation characteristic receive_type. In receive_type you define whether the Receive and Receive_Mapped_Data calls are to be executed with blocking or without. The call overwrites the value of receive_type which was assigned during the Initialize_Conversation call.

The Set_Receive_Type call is only permitted in one of the following states: “Initialize”, “Send” or “Receive”.

This function is one of the advanced functions.

UPIC local:
Local connection via UPIC local does not support the Set_Receive_Type call.

Syntax

CMSRT (conversation_ID, receive_type, return_code)

Parameters

→ conversation_ID Identifier of the conversation
→ receive_type Defines whether the following Receive / Receive_Mapped_Data calls are to be executed with blocking or without. You can specify the following values:

  – CM_RECEIVE_AND_WAIT
The Receive and Receive_Mapped_Data calls have a blocking effect, i.e. if no information is available at the time of the call, the program run waits until information arrives for this conversation. Only then does the program run return from the Receive or Receive_Mapped_Data call and transfer the data to the program. If there is information available at the time of the call, the program receives it without waiting.
If a maximum wait time (timeout timer) was set with Set_Receive_Timer before the Receive or Receive_Mapped_Data call, the program run returns from the Receive or Receive_Mapped_Data call on expiry of this wait time, even if there is still no information available.
Set_Receive_Type

CM_RECEIVE_IMMEDIATE
The Receive and Receive_Mapped_Data calls have a non-blocking effect, i.e. if there is information present at the time of the call, the program receives it without waiting. If there is no information at the time of the call, the program does not wait. The program run returns from the Receive or Receive_Mapped_Data call immediately.

← return_code Result of the function call.

Result (return_code)
CM_OK The call is OK.
CM_PROGRAM_PARAMETER_CHECK
conversation_ID is invalid or the value of receive_type is undefined.
CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.
CM_CALL_NOT_SUPPORTED
The function is not supported.

State change
If there are no errors, this function returns the result CM_OK. This call does not change the state of the conversation.

Notes
● If the return code is not CM_OK, the receive_type characteristic remains unchanged.
● If Set_Receive_Type is called in the “Start” or “Reset” state, the value transferred in conversation_ID is always invalid. The return code CM_PROGRAM_PARAMETER_CHECK is then always returned as the result of the call.
Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers.
Check whether the memory requirements of your program are too high and if
necessary reboot your system.

CM_CALL_NOT_SUPPORTED
This is not necessarily an error: If the application is intended for both UPIC-L and
UPIC-R this return code just means that the application is linked to a UPIC-L library.
The program can take note of this return code and avoid making further
Set_Receive_Type calls.

Function declaration: Set_Receive_Type

CM_ENTRY Set_Receive_Type ( unsigned char CM_PTR conversation_ID,
                        CM_RECEIVE_TYPE CM_PTR receive_type,
                        CM_RETURN_CODE CM_PTR return_code )
Set_Sync_Level - Setting a synchronization level

The Set_Sync_Level (CMSSL) call sets the value for the sync_level conversation characteristic. The call overwrites the value that was assigned at the Initialize_Conversation call.

The Set_Sync_Level call cannot be executed after an Allocate call.

This function is one of the advanced functions.

Syntax

CMSSL (conversation_ID, sync_level, return_code)

Parameters

→ conversation_ID Identifier of the conversation.
→ sync_level Defines the level of synchronization that the local CPI-C program and the remote UTM application can use during this conversation.
  sync_level must have the value CM_NONE.
← return_code Result of the function call.

Result (return_code)

CM_OK
  The call is OK.
CM_PROGRAM_STATE_CHECK
  The conversation is not in the “Initialize” state.
CM_PROGRAM_PARAMETER_CHECK
  conversation_ID is invalid or the value in sync_level is undefined.
CM_PRODUCT_SPECIFIC_ERROR
  The UPIC instance could not be found.

State change

If there are no errors, this function returns the result CM_OK. This call does not change the state of the conversation.
Set_Sync_Level

Note

The call serves only to improve the portability of CPI-C programs. Even if it returns CM_OK, \( \text{sync\textunderscore\_level} \) is not changed. UPIC internally always uses "\( \text{sync\textunderscore\_level}=\text{CM\textunderscore\_NONE} \)".

Behavior in the event of errors

\textbf{CM\_PROGRAM\_STATE\_CHECK}
Modify program.

\textbf{CM\_PROGRAM\_PARAMETER\_CHECK}
Modify program.

\textbf{CM\_PRODUCT\_SPECIFIC\_ERROR}
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

Function declaration: Set_Sync_Level

\texttt{CM\_ENTRY Set\_Sync\_Level ( \texttt{unsigned char CM\_PTR conversation\_ID,} \texttt{CM\_SYNC\_LEVEL CM\_PTR sync\_level,} \texttt{CM\_RETURN\_CODE CM\_PTR return\_code})}
**Set_TP_Name - Setting TP-name**

A program uses the `Set_TP_Name` (CMSTPN) call to set the values of the conversation characteristics `TP_name` and `TP_name_length`. The `TP_name` is the transaction code of a UTM program unit.

`Set_TP_Name` modifies the values taken from the side information with the `Initialize_Conversation` call. The modified values apply only for the duration of a conversation; the values in the side information itself remain unchanged.

The `Set_TP_Name` call cannot be executed after the `Allocate` call.

This call is one of the advanced functions.

**Syntax**

CMSTPN (conversation_ID, TP_name, TP_name_length, return_code)

**Parameters**

- → conversation_ID  Identifier of the conversation.
- → TP_name  UTM transaction code.
- → TP_name_length  Length of `TP_name`.
  Minimum: 1, maximum: 8
- ← return_code  Result of the function call.

**Result (return_code)**

- CM_OK  The call is OK.
- CM_PROGRAM_STATE_CHECK  The call is not permitted in this state.
- CM_PROGRAM_PARAMETER_CHECK  The `conversation_ID` or `TP_name` is invalid or the value in `TP_name_length` is less than 1 or greater than 8.
- CM_PRODUCT_SPECIFIC_ERROR  The UPIC instance could not be found.

If the return code is not CM_OK, `TP_name` and `TP_name_length` remain unchanged.
State change

None

Behavior in the event of errors:

CM_PROGRAM_STATE_CHECK
  Modify program.

CM_PROGRAM_PARAMETER_CHECK
  Modify program.

CM_PRODUCT_SPECIFIC_ERROR
  The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

Function declaration: Set_TP_Name

CM_ENTRY Set_TP_name ( unsigned char CM_PTR conversation_ID,
                      unsigned char CM_PTR TP_name,
                      CM_INT32 CM_PTR TP_name_length,
                      CM_RETURN_CODE CM_PTR return_code )
Specify_Local_Port - Setting the TCP/IP port of the local application

The Specify_Local_Port (CMSLP) call sets the port number of the local application. The call overwrites the value assigned using the Enable_UTM_UPIC call. After the Initialize_Conversation call, this call may no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

UPIC-Local:
Connection via UPIC local does not support the Specify_Local_Port call.

Syntax
CMSLP (port_number, return_code)

Parameters

→ port_number Specifies which port number the local application uses when signing on to the communication system.
Minimum: 0, maximum: 32767

← return_code Result of the function call

Result (return_code)

CM_OK
The call is OK.

CM_CALL_NOT_SUPPORTED
The function is not supported. This return code occurs in UPIC-L and in UPIC-R on BS2000/OSD.

This code is always returned with UPIC-L. It indicates to the program that a port number cannot be assigned because UPIC-L does not require this information as a result of the underlying communication system.

The code is only returned with UPIC-R (BS2000) in the event that the UPIC library on BS2000 is used together with CMX. The CMX communication system used by UPIC-R does not provide any option on BS2000 systems for passing IP addresses for addressing the partner application at the interface. If the UPIC library uses the Socket interface as its communication system, the code is never returned.

CM_PROGRAM_STATE_CHECK
The conversation is not in the “Reset” state.
Specify_Local_Port

CM_PRODUCT_SPECIFIC_ERROR
The UPIC instance could not be found.

CM_PROGRAM_PARAMETER_CHECK
The value of port_number is invalid.

State change
The call does not change the state of the conversation.

Note
The local port number is a purely formal value which has no effect whatsoever. Specification of this value is only supported for reasons of compatibility. It should be omitted.

Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
Modify program.

CM_PROGRAM_STATE_CHECK
Modify program.

CM_PRODUCT_SPECIFIC_ERROR
The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

CM_CALL_NOT_SUPPORTED
This is not necessarily an error: If the application is intended for both UPIC-L and UPIC-R on Unix systems and Windows systems this return code just means that the application is linked to a UPIC-L library. The program can take note of this return code and avoid making further calls to set address information.

On BS2000/OSD, this return code means that the application is connected to UPIC-R and CMX. The program can remember this return code and then no longer requires the Specify_Local_Port call.

Function declaration: Specify_Local_Port

CM_ENTRY Specify_Local_Port ( CM_INT32 CM_PTR port_number,
CM_RETURN_CODE CM_PTR return_code )
Specify_Local_Tsel - Setting the T-SEL of the local application

The `Specify_Local_Tsel` (CMSLT) call sets the value of the T-SEL characteristic of the local application. The call overwrites the value assigned using the `Enable_UTM_UPIC` call. After the `Initialize_Conversation` call, this call may no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

XPW UPIC-Local:

Connection via UPIC local does not support the `Specify_Local_Tsel` call.

**Syntax**

CMSLT (transport_selector, transport_selector_length, return_code)

**Parameters**

- transport_selector  
  Transport selector of the local application which is transferred to the communication system

- transport_selector_length  
  Length of the transport selector in bytes.
  Minimum: 0, maximum: 8
  If the length of the transport selector is entered as 0, the name of the local application itself is used as the transport selector.

- return_code  
  Result of the function call.
Result *(return_code)*

**CM_OK**

The call is OK.

**CM_CALL_NOT_SUPPORTED**

The function is not supported. This return code only occurs in UPIC-L. It indicates to the program that a T-SEL cannot be assigned because UPIC-L does not require this information because of the underlying communication system.

**CM_PROGRAM_STATE_CHECK**

The conversation is not in the “Reset” state.

**CM_PRODUCT_SPECIFIC_ERROR**

The UPIC instance could not be found.

**CM_PROGRAM_PARAMETER_CHECK**

The value of transport_selector_length is invalid.

State change

The call does not change the state of the conversation.

Behavior in the event of errors

**CM_CALL_NOT_SUPPORTED**

Is not necessarily an error: If an application is intended for both UPIC-L and UPIC-R, this return code just means that the application is linked to a UPIC-L library. The program can take note of this return code and avoid sending further calls to set address information.

**CM_PROGRAM_PARAMETER_CHECK**

Modify program.

**CM_PROGRAM_STATE_CHECK**

Modify program.

**CM_PRODUCT_SPECIFIC_ERROR**

The operating system cannot provide sufficient memory for the internal buffers. Check whether the memory requirements of your program are too high and if necessary reboot your system.

Function declaration: *Specify_Local_Tsel*

```c
CM_ENTRY Specify_Local_Tsel (unsigned char CM_PTR transport_selector,
                          CM_INT32 CM_PTR transport_selector_length,
                          CM_RETURN_CODE CM_PTR return_code )
```
Specify_Local_Tsel_Format - Setting the TSEL format of the local application

The **Specify_Local_Tsel_Format** (CMSLTF) call sets the value of the *T-SEL-FORMAT* characteristic of the local application. The call overwrites the value assigned by the *Enable_UTM_UPIC* call. After the *Initialize_Conversation* call, this call may no longer be issued.

This function is one of the additional functions of the UPIC carrier system; it is not a component of the CPI-C interface.

**UPIC-Local:**

Connection via UPIC local does not support the *Specify_Local_Tsel_Format* call.

**Syntax**

CMSLTF (tsel_format, return_code)

**Parameters**

→ tsel_format Specifies which character set is to be used for the transport selector (TSEL). The following values can be entered:

– **CM_TRANSDATA_FORMAT**
  The transport selector is transferred to the communication system using TRANSDATA format.

– **CM_EBCDIC_FORMAT**
  The transport selector is transferred to the communication system using EBCDIC format.

– **CM_ASCII_FORMAT**
  The transport selector is transferred to the communication system using ASCII format.

← return_code Result of the function call.
Result (return_code)

CM_OK
   The call is OK.

CM_CALL_NOT_SUPPORTED
   The function is not supported. This return code only occurs in UPIC-L. It indicates
to the program that a format cannot be assigned for the transport selector because
UPIC-L does not require this information as a result of the underlying communi-
cation system.

CM_PROGRAM_STATE_CHECK
   The conversation is not in the “Reset” state.

CM_PRODUCT_SPECIFIC_ERROR
   The UPIC instance could not be found.

CM_PROGRAM_PARAMETER_CHECK
   The value of tsel_format is invalid.

State change

The call does not change the state of the conversation.

Behavior in the event of errors

CM_PROGRAM_PARAMETER_CHECK
   Modify program.

CM_PROGRAM_STATE_CHECK
   Modify program.

CM_PRODUCT_SPECIFIC_ERROR
   The operating system cannot provide sufficient memory for the internal buffer.
   Check whether the memory requirements of your program is too high and if
   necessary reboot your system.

CM_CALL_NOT_SUPPORTED
   Is not necessarily an error: If an application is intended for both UPIC-L and
   UPIC-R, this return code just means that the application is linked to a UPIC-L library.
   The program can take note of this return code and avoid sending further calls to set
   address information.

Function declaration: Specify_Local_Tsel_Format

CM_ENTRY Specify_Local_Tsel_Format ( CM_TSEL_FORMAT CM_PTR tsel_format,
                                    CM_RETURN_CODE CM_PTR return_code )
Specify_Secondary_Return_Code - Setting the properties of the secondary return code

The Specify_Secondary_Return_Code (CMSSRC) call causes the program to set the secondary return code property of the CPI-C calls.

This function belongs to the additional UPIC carrier system functions; it is not a component of the CPI-C interface.

Syntax

CMSSRC (return_type, return_code)

Parameters

→ return_type Specifies the secondary return code property of the CPI-C calls. The following values can be specified:

CM_RETURN_TYPE_PRIMARY: The corresponding UPIC calls return the secondary return code.
CM_RETURN_TYPE_SECONDARY: The secondary return code can be read out only by means of the CMESRC call. The corresponding UPIC calls do not return a secondary return code.

← return_code Result of the function call.

Result (return_code)

CM_OK The call is OK
CM_NO_SECONDARY_RETURN_CODE The secondary return code property is not available.
CM_PROGRAM_PARAMETER_CHECK The value of return_type is invalid.
CM_PROGRAM_STATE_CHECK The program is in the “Start” state.
CM_PRODUCT_SPECIFIC_ERROR The UPIC instance could not be found.
**Note**

The function can be called directly after an `Enable_UTM_UPIC` call. It has no effect on the `Enable_UTM_UPIC` call.

**State change**

No state change.

**Behavior in the event of errors**

- **CM_PROGRAM_PARAMETER_CHECK**
  
  Modify program.

- **CM_PROGRAM_STATE_CHECK**
  
  Modify program.

- **CM_PRODUCT_SPECIFIC_ERROR**
  
  The operating system cannot provide sufficient memory for internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

- **CM_NO_SECONDARY_RETURN_CODE**
  
  Is not necessarily an error. If a UPIC-R application communicates with various UTM partners, some of which can support secondary return codes and some of which cannot, this return code means simply that the application wishes to communicate with a UTM application that does not support secondary return codes. The program can take note of this return code and dispense with further `Extract_Secondary_Return_Code` calls.

**Function declaration: Specify_Secondary_Return_Code**

```c
CM_ENTRY Specify_Secondary_Return_Code (  
  CM_INT32        CM_PTR  return_type,  
  CM_RETURN_CODE  CM_PTR  return_code )
```
5.10 COBOL interface

The CPI-C-COBOL program interface largely corresponds to the C interface described in section 5.9 on page 132ff. You can therefore consult this description when creating CPI-C programs in COBOL. This section groups together the special features of the COBOL interface which apply for the data structures and CPI-C calls.

COPY element CMCOBOL

The COPY element CMCOBOL, which contains the condition variables and names, is supplied for CPI-C applications in COBOL. After installation of the UPIC carrier system, you will find CMCOBOL

- under Window systems in the file `upic_installation_directory\copy-cob`
- in Unix systems in the file `upic_installation_directory/copy-cobol85`
- on BS2000/OSD systems in the library `SYSLIB.UTM-CLIENT.060`.

CMCOBOL must be copied into the WORKING-STORAGE-SECTION using the COPY statement. The names of constants are distinguished from the C names only through the use of hyphens instead of underscores, e.g. “CM-SEND-RECEIVED” instead of “CM_SEND_RECEIVED”.

The name TIME-OUT or TIMEOUT is used in CMCOBOL for the CPI-C interface as a result of the CPI-C specification. These words are reserved by Micro Focus, so this name must be modified in the source, for example using the following statement:

```
COPY CMCOBOL REPLACING TIME-OUT BY CPIC-TIMEOUT
```

CPI-C calls in COBOL

The function names are identical in C and COBOL. The following applies for the parameters of the CPI-C calls:

- As is normal in COBOL, the parameters must be transferred by reference.
- Each variable in the parameter list must begin with the level number 01.
- Numerical data must be in the COMP format that produces the same binary format as with C on the respective machine.

- When using COBOL under Windows systems you must bear in mind the necessary call conventions for the dynamic library (DLL).
Example

Extract from a program with the “Initialize” call:

```cobol
... WORKING-STORAGE-SECTION.
*************************
COPY CMCOBOL.
...
PROCEDURE DIVISION.
*************************
...
CALL "CMINIT" USING CONVERSATION-ID,SYM-DEST-NAME,CM-RETCODE.
```
6 XATMI interface

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

The XATMI program interface is based on the X/Open CAE Specification “Distributed Transaction Processing: The XATMI Specification” of November 1995. Knowledge of this specification is essential for understanding this chapter. This chapter describes the XATMI interface for openUTM clients using UPIC.

For information on OpenCPIC, please refer to the manual “openUTM-Client for the OpenCPIC Carrier System”.

With a few exceptions, the description of the XATMI interface is platform-independent. The exceptions are indicated in the text.

Terms

The following terms are used in this description:

Service A service function that is programmed in C or COBOL in accordance with the XATMI specification. XATMI distinguishes between two different types of services: end services and intermediate services.
– An “end” service is linked only to its client and does not call any other services.
– An “intermediate” service calls one or more other services.

Client An application that calls service functions.

Server A UTM application containing the service functions in C and/or COBOL. The service functions can comprise a number of program units.

Request A request is a service call. This call can be initiated by a client or by an intermediate service.

Requester The XATMI specification uses the term “requester” to refer to the application that calls a service. A requester can be either a client or a server.

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

The diagram below shows the connection of client/server applications, linking the client, server and requester. They exchange their type-encoded data structures (typed buffers) in accordance with the protocol of the “XATMI U-ASE Definition”.

Figure 16: Client/server applications

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

– in the case of a server, all available services
– in the case of a client, the services of all servers to which the client is connected
– in the case of a requester, all services available as well as all services used

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

A number of communication paradigms are available for processing the client/server connections Con11, Con13,... (see section “Communication paradigms” on page 278).
6.1.1 Default server

To simplify the client/server configuration openUTM client allows you to declare a default server using the statement DEST=.DEFAULT in the SVCU statement of the local configuration file (see page 295).

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

\[\text{SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=.DEFAULT, MODE=RR}\]

In this case UPIC expects a suitable default server entry in the upicfile, i.e.

\[\text{LN.DEFAULT localname}\\\text{SD.DEFAULT servername}\]

Furthermore you are allowed to call a service svcname2@BRANCH9 using DEST=BRANCH9 without entering a declaration in the local configuration file. In such a case the following entry is assumed:

\[\text{SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=BRANCH9, MODE=RR}\]

The partner, in this case BRANCH9, must be known to the carrier system. However, if the local configuration file contains an entry svcname2@BRANCH9, this entry takes precedence over the default server assumption.

6.1.2 Restart

Although there is no service restart for XATMI (as XATMI does not support complex services), you have the option of defining a recovery service, which resends the last output message from openUTM to the client.

This recovery service is defined with the transaction code KDCRECVR.
6.2 Communication paradigms

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

- synchronous request response paradigm: single-step dialog.
  The client is blocked after sending the service request until it receives a response.

- asynchronous request response paradigm: single-step dialog.
  The client is not blocked after sending the service request.

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

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

**Synchronous request-response paradigm**

The client only needs one single `tpcall()` call for the communication.

The `tpcall()` call addresses the service, sends precisely one message to this service, and waits until it receives a response, i.e. `tpcall()` has a blocking effect.

![Figure 17: Synchronous request response paradigm](image)

In this diagram, `svc` is the internally used service name, `svcinfo` is the service info structure with the service name, and `tpservice` is the program name of the service routine. The service info structure is part of the XATMI interface.

With this paradigm, a dialog TAC for the requested service has to be generated on the UTM server side.
Asynchronous request-response paradigm

With this paradigm, communication is handled in two steps. In the first step, a `tpacall()` call is used to address the service and send the message. In the second step the response is fetched with `tpgetrply()` at a later stage, see diagram below.

![Diagram of Asynchronous Request-Response Paradigm](image)

Figure 18: Asynchronous request response paradigm

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

`tpacall` is non-blocking, i.e. the client can carry out other local processing tasks in the meantime. However, the client cannot call another service, as only one job is permitted at any one time with the UPIC carrier system.

If the client is to engage several services in parallel, you must use the OpenCPIC carrier system.

In contrast, `tpgetrply` is blocking, which means that the client waits until the response is received.

With this paradigm, a dialog TAC must be generated for the service on the UTM server side (as with synchronous request-response).
Conversational paradigm

XATMI offers the conversational paradigm for connection-oriented tasks (“conversations”).

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

In the conversational paradigm, the conversation is explicitly established to a service with the *tpconnect* call. As long as the conversation exists, the client and server can exchange data with *tpsend* and *tprecv*. However, this “dialog” is not a dialog in the sense of OSI TP, and only one transaction can be processed.

The conversation is terminated when the server signals the end with *tpreturn*; the client then receives a corresponding code with *tprecv* in the *tperrno* variable. The client program must therefore contain at least one *tprecv* call.

![Figure 19: Conversational paradigm](image)

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

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

In the event of errors, the client can force a conversation abort with the *tpdiscon* call.
6.3 Typed buffers

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

The advantage of this is that the application need not take account of any machine dependencies, such as Big Endian/Little Endian representation, ASCII/EBCDIC conversion, or alignment with word limits. This means that data types such as int, long, float, etc. can be transferred as such. There is no need for any encoding/decoding by the application because this is carried out by XATMI (in accordance with the rules of the XATMI U-ASE definition).

A data buffer object comprises four components:

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

This type of data buffer is created at runtime and can then be addressed by its variable name (= subtype name). The subtype defines the structure, while the type defines the set of values of the permitted elementary data types. In C programs, these buffers are created dynamically with \texttt{tpallcoc} and are then called “typed buffers”; in COBOL programs, these buffers are defined statically and are then called “typed records”.

Types

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

Three types are defined in XATMI:

\textbf{X\_OCTET} \quad Non-typed data stream of bytes (“user buffer”). This type has no subtypes. No conversion takes place.

\textbf{X\_COMMON} \quad All data types that can be used in common by C and COBOL. Conversion is carried out by XATMI.

\textbf{X\_C\_TYPE} \quad All elementary C data types, with the exception of pointers. Conversion is carried out by XATMI.
Subtypes

A subtype has a name of up to 16 characters, with which it is addressed in the application program. Each subtype is assigned a data structure (C structure or COBOL record) which determines the syntax of the subtype, see page 292.

The data types must not be nested.

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

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>ASN.1 type</th>
<th>X_C_TYPE</th>
<th>X_COMMON</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>short integer</td>
<td>INTEGER</td>
<td>short</td>
<td>S9(4) COMP-5</td>
</tr>
<tr>
<td>S&lt;n&gt;</td>
<td>short integer array</td>
<td>SEQUENCE OF INTEGER</td>
<td>short[n]</td>
<td>S9(4) COMP-5 ...</td>
</tr>
<tr>
<td>i</td>
<td>integer</td>
<td>INTEGER</td>
<td>integer</td>
<td>--²</td>
</tr>
<tr>
<td>I&lt;n&gt;</td>
<td>integer array</td>
<td>SEQUENCE OF INTEGER</td>
<td>integer[n]</td>
<td>--</td>
</tr>
<tr>
<td>l</td>
<td>long integer</td>
<td>INTEGER</td>
<td>long</td>
<td>S9(9) COMP-5</td>
</tr>
<tr>
<td>L&lt;n&gt;</td>
<td>long integer array</td>
<td>SEQUENCE OF INTEGER</td>
<td>long[n]</td>
<td>S9(9) COMP-5 ...</td>
</tr>
<tr>
<td>f</td>
<td>float</td>
<td>REAL</td>
<td>float</td>
<td>--</td>
</tr>
<tr>
<td>F&lt;n&gt;</td>
<td>float array</td>
<td>SEQUENCE OF REAL</td>
<td>float[n]</td>
<td>--</td>
</tr>
<tr>
<td>d</td>
<td>double</td>
<td>REAL</td>
<td>double</td>
<td>--</td>
</tr>
<tr>
<td>D&lt;n&gt;</td>
<td>double array</td>
<td>SEQUENCE OF REAL</td>
<td>double[n]</td>
<td>--</td>
</tr>
<tr>
<td>c</td>
<td>character</td>
<td>OCTET STRING</td>
<td>char</td>
<td>PIC X</td>
</tr>
<tr>
<td>t</td>
<td>character</td>
<td>T.61-String</td>
<td>char</td>
<td>PIC X</td>
</tr>
<tr>
<td>C&lt;n&gt;</td>
<td>character array: All values from 0 thru 255 (decimal)</td>
<td>OCTET STRING</td>
<td>char[n]</td>
<td>PIC X(n)</td>
</tr>
<tr>
<td>C&lt;!n&gt;</td>
<td>character array, terminated by null ('\0')</td>
<td>OCTET STRING</td>
<td>char[n]</td>
<td>--</td>
</tr>
<tr>
<td>C&lt;m&gt;:&lt;n&gt;</td>
<td>character matrix⁴</td>
<td>SEQUENCE OF OCTET STRING</td>
<td>char [m][n]</td>
<td>--</td>
</tr>
<tr>
<td>C&lt;!m&gt;:&lt;n&gt;</td>
<td>character matrix, terminated by null ('\0')</td>
<td>SEQUENCE OF OCTET STRING</td>
<td>char [m][n]</td>
<td>--</td>
</tr>
<tr>
<td>T&lt;n&gt;</td>
<td>The printable characters A-Z, a-z, and 0-9 plus a range of special characters and control characters, see page 384.</td>
<td>T.61 string</td>
<td>char[n]</td>
<td>PIC X(n)</td>
</tr>
</tbody>
</table>
The assignment between data structures, subtypes, and desired services is defined in the local configuration, see page 295.

**Character set conversion with X_C_TYPE and X_COMMON**

The data buffers are transmitted over the network encoded in the ASCII character set. However, a partner can use a different character set encoding instead of ASCII, for example a BS2000 application which uses EBCDIC. In this case, the XATMI library converts the ASN.1-type T.61 string for all incoming and outgoing data (with the exception that OCTET strings are not converted).

Therefore no automatic conversion may be generated. For the UPIC carrier system this means the respective identifier must be generated in the upicfile:

- **X/W** – This is SD for Unix systems and Windows systems.
- **B** – This is HD for BS2000/OSD.
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 can be found in the X/Open Specification “Distributed Transaction Processing: The XATMI Specification”. Knowledge of this specification is essential for creating XATMI programs.

The program interface is available for both C and COBOL.

6.4.1 XATMI functions for clients

The tables below list all XATMI client calls and indicate the communication paradigm with which they can be used and if the function may also be called by a server.

In addition there are the UTM-Client calls \textit{tpinit} and \textit{tpterm}. These two functions are not included in the XATMI standard and are used to connect XATMI to the carrier system. They are described in section “Calls for connecting to the carrier system” on page 285.

### Calls of the request/response paradigm

<table>
<thead>
<tr>
<th>C call</th>
<th>COBOL call</th>
<th>Call in Client/Server</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpcall</td>
<td>TPCALL</td>
<td>C</td>
<td>Service request in synchronous request/response paradigm</td>
</tr>
<tr>
<td>tpacall</td>
<td>TPACALL</td>
<td>C</td>
<td>Service request in asynchronous request/response paradigm or single request paradigm (flag TPNOREPLY set)</td>
</tr>
<tr>
<td>tpgetrply</td>
<td>TPGETRPLY</td>
<td>C</td>
<td>Response request in synchronous request/response paradigm</td>
</tr>
<tr>
<td>tpcancel</td>
<td>TPCANCEL</td>
<td>C</td>
<td>Deletes an asynchronous service request before the requested response is received</td>
</tr>
</tbody>
</table>

Table 11: Calls of the request/response paradigm
Calls for the conversational paradigm

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

Table 12: Calls for the conversational paradigm

Calls for typed buffers

<table>
<thead>
<tr>
<th>C call</th>
<th>COBOL call</th>
<th>Call in Client/Server</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpalloc</td>
<td>--</td>
<td>C, S</td>
<td>reserves memory area for a typed buffer</td>
</tr>
<tr>
<td>tpdealloc</td>
<td>--</td>
<td>C, S</td>
<td>modifies the size of a typed buffer</td>
</tr>
<tr>
<td>tpfree</td>
<td>--</td>
<td>C, S</td>
<td>releases a typed buffer</td>
</tr>
<tr>
<td>tptypes</td>
<td>--</td>
<td>C, S</td>
<td>ascertains the type of a typed buffer</td>
</tr>
</tbody>
</table>

Table 13: Calls for typed buffers

6.4.2 Calls for connecting to the carrier system

The openUTM client may use UPIC or OpenCPIC as the carrier system. An XATMI application program must therefore explicitly sign on to the carrier system using `tpinit` and sign off using `tpterm`, i.e. the program has the following structure:

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

The two calls `tpinit` and `tpterm` are described below.

For a general description of the UTM user concept, see section “User concept, security and restart” on page 113.
tpinit - Initializing the client

Format

C:      #include <xatmi.h>
       int tpinit (TPCLTINFO *tpinfo)           (in)

COBOL:  01 TPCLTDEF-REC.
        COPY TPCLTDEF.
        CALL "TPINIT" USING TPCLTDEF-REC.

Description

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

C structure TPCLTINFO:
#define MAXTIDENT 9
typedef struct {
   long flags;                    /* for future use */
   char usrrname[MAXTIDENT];
   char cltname[MAXTIDENT];
   char passwd [MAXTIDENT];
} TPCLTINFO;

COBOL record TPCLTDEF:
05 FLAG           PIC S(9) COMP-5.
05 USRNAME        PIC X(9).
05 CLTNAME        PIC X(9).
05 PASSWD         PIC X(9).

A user ID is entered in usrrname and a password in passwd. Both parameters are used to establish a conversation, and serve as proof of access authorization on the UTM side. cltname (= local client name) identifies the client to the carrier system.

cltname is

X/W – With UPIC-L it is the PTERM name or the local application name from the upicfile
 – With UPIC-R it is the the upicfile entry or TNS entry (Unix system or Windows system, see section “Configuration with TNS entries” on page 322) or the BCMAP entry (BS2000/OSD, see section “Configuration using BCMAP entries” on page 322).
If \textit{username} and \textit{passwd} are initialized with the null string (COBOL: spaces), the security functions are not activated, i.e. there is no access control in openUTM. If at least one of these two parameters contains a valid value, this is checked by UTM.

If \textit{cltname} is initialized with the null string or with spaces, the local client name is preset to 8 spaces.

In C, if \textit{tpinit} is called with a null pointer, then no access control is activated and the local client name is preset to 8 blanks. In COBOL, the structure must be filled with spaces for this purpose.

The entries in \textit{username}, \textit{passwd}, and \textit{cltname} (if any) must comply with the UTM name conventions, i.e. they can be up to eight characters in length and, in C, must be terminated with the end-of-string character ("\0").

\textbf{Return codes}

In the event of an error, \textit{tpinit} returns the value -1 and sets the \textit{tperrno} error variable to one of the following values:

- \textbf{TPEINVAL}
  - One or more parameters were assigned invalid values.

- \textbf{TPENOENT}
  - Initialization could not be performed, e.g. there may not be sufficient memory for internal buffers.

- \textbf{TPEPROTO}
  - \textit{tpinit} was called at an inappropriate time, e.g. the client is already initialized.

- \textbf{TPESYSTEM}
  - An internal error has occurred.
**tpterm - Signing the client off**

**Format**

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

**Description**

The `tpterm` function is used to sign a client off from the carrier system. The client is the one in which this function is called and must have been initialized previously with `tpinit`. Following a `tpterm` call, no further XATMI calls (apart from `tpinit`) are permitted.

**Return codes**

In the event of an error, `tpterm` returns the value -1 and sets the `tperrno` error variable to one of the following values:

TPENOENT
   The client could not sign off in the normal way. There may be problems in the carrier system.

TPEPROTO
   `tpterm` was called at an inappropriate time, i.e. the client is not yet initialized.

TPESYSTEM
   An internal error has occurred.
6.4.3 Transaction control

When an XATMI service is called, the client uses the call parameter flag (in C) or the TPTRAN-FLAG (in COBOL) to control whether or not a called UTM service is included in the global transaction.

The XATMI-C interface includes the service in the global transaction by default. In order to exclude the service from the global transaction, you must set the TPNOTRAN flag explicitly.

No default value exists for the XATMI-COBOL interface, you must set either TPTRAN or TPNOTRAN.

If the service is started with the TPTRAN flag, then it is included in the global transaction.

When using the tpreturn() call, the parameter `rval` returns the values TPSUCCESS or TPFAIL. This determines whether the transaction is terminated successfully or reset.

When using the XATMI interface with the UPIC carrier system the TPTRAN flag is ignored and the TPNOTRAN flag set internally. This behaviour improves the portability of XATMI programs.

6.4.4 Mixed operation

Mixed operation refers to communication between an XATMI program and a CPI-C program.

For interaction with a CPI-C program the XATMI program must contain the corresponding CPI-C calls, although the connection is established by the XATMI partner. For communication with a partner, the same interface must be used on both sides, i.e. the `Deallocate()` call is forbidden in XATMI programs.

6.4.5 Administration interface

In XATMI programs, only the KDCS call KDCADMI() can be used; other KDCS calls are not permitted.

On the UTM side, the corresponding TAC and possibly USER must be generated with administration authorization during KDCDEF generation.
6.4.6 Header files and COPY elements

For the creation of openUTM-Client programs which use the XATMI interface, header files for C and COPY elements for COBOL are supplied.

When linking the client programs, the UTM client library must be incorporated.

C modules with XATMI calls require the following files:

1. The header file xatmi.h.
2. The file(s) with the data structures for all typed buffers used in the module, see also page 281.

COBOL modules with XATMI calls require the following COPY elements and files:

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

In Windows systems the XATMI interface is not supported in COBOL.

Windows systems

Under Windows systems you will find the header files in the directory

\xatmipath\include

where \xatmipath is the directory in which XATMI is installed. The directory C:\Programme\xatmi is set as default.

No COPY elements are supplied for COBOL.

Unix systems

Under Unix systems you will find the header files in the directory

\upicpath/xatmi/include

and the COPY elements in the directory

\upicpath/xatmi/copy-cobol85
The openUTM client library is called
`upicpath/sys/libxtclt.a` or `upicpath/sys/libxtclt.so` (shared objects)

where `upicpath` is the directory in which openUTM-Client was installed.

**BS2000/OSD**

Under BS2000/OSD the include files and the COPY members are S type members of the library

`$userid.SYSLIB.UTM-CLIENT.060`

where `$userid` stands for the ID under which the openUTM client was installed.

### 6.4.7 Events and error handling

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

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

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

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

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

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

Data is exchanged between the programs on the basis of these data structures, which must therefore be known to both the client and the server. All data types described in the table on page 282 are permitted.

The header files or COBOL COPY files in which the typed buffers are described serve as input for the generation program xatmigen or xtgen32, see page 300. The following rules apply to these files:

– C and COBOL data structures must be contained in separate files. A file that contains both C includes and COBOL COPY elements is not permitted as input.

– The files can only comprise definitions of data structures, blank lines, and comment statements. Macro statements, i.e. statements beginning with ‘#’, are permitted in C.

– The data structure definitions must be specified in full. In particular, COBOL data records must begin with the level number “01”.

– The data structures must not be nested.

– Only absolute values are permitted as field lengths, macro constants are not accepted.

– Only the data types listed in the table on page 282 are permitted. In particular, no pointer types are permitted in C.

The user may have to use the generation tool xatmigen or xtgen32 to map the character arrays to ASN.1 string types because neither C nor COBOL recognizes these data types; see section “xatmigen or xtgen32 tool” on page 300.

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

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

1. C include for typed buffer

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

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

2. COBOL COPY for typed record

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

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

Further examples can be found in the X/Open Specification on XATMI.
6.4.9 Characteristics of XATMI in UPIC

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

- All XATMI calls relevant for clients are supported. Additionally the two calls `tpinit` and `tpterm` are provided.
- Only one conversation per service is allowed.
- A maximum of 100 buffer entities can be used simultaneously within a client application. For example, with an application in C this is 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 less than the maximum possible message length because the messages contain an “overhead” in addition to the net data. The more complex the buffer, the bigger the overhead.
  
  The following applies as a rule of thumb: max. buffer size = 2/3 of max. message length

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

- The following limits apply to name lengths:
  
  service name  16 bytes
  buffer name    16 bytes

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

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

6.5.1 Creating the local configuration file

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

- A line begins with an SVCU or BUFFER statement and specifies precisely one service or one subtype (=typed buffer).
- Two operands are separated by a comma.
- A statement is concluded by a semicolon (‘;’).
- If the statement occupies more than one line, the continuation character ‘\’ (backslash) must appear at the end of each line.
- A comment line begins with the ‘#’ character.
- Blank lines can be inserted, e.g. to improve legibility.

Using the xatmigen tool, you create the actual local configuration file (page 301) from the file which contains the local configuration definition.

The SVCU and BUFFER statements are described below.
**SVCU statement: Define available service**

In an SVCU statement, the characteristics required to call a service in the partner application are described for the client.

The SVCU statement can be omitted, if a default server is declared in the side information file of UPIC (*upicfile*) with *transaction-code* = *remote-service-name* = *internal-service-name*.

*Default-Server:*

To simplify the client/server configuration openUTM client allows you to declare a default server using the statement *DEST=.DEFAULT* in the SVCU statement of the local configuration file (see page 295).

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

```
SVCU svcname2, RSN=svcname2, TAC=SCVname2, DEST=.DEFAULT, MODE=RR
```

*In this case UPIC expects a suitable default server entry in the *upicfile*, i.e.*

*LN.DEFAULT localname*

*SD.DEFAULT servername*

*Furthermore you are allowed to call a service *svcname2@BRANCH9* using *DEST=BRANCH9* without entering a declaration in the local configuration file. In such a case the following entry is assumed:*

```
SVCU svcname2, RSN=svcname2, TAC=SCVname2, DEST=BRANCH9, MODE=RR
```
The partner, in this case BRANCH9, must be known to UPIC. However, if the local configuration file contains an entry `svcname2@BRANCH9`, this entry will be used.

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

**internal-service-name**
A name of up to 16 bytes under which a (remote) service can be addressed in the program. This name must be unique within the application, i.e. it can only appear once in the LCF.

Mandatory operand!

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

If this operand is omitted, `xatmigen` sets `RSN=internal-service-name`.

**TAC=transaction-code**
A transaction code of up to 8 bytes with which the service must be generated in the remote application.

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

The transaction code KDCRECVR can be used to define a recovery service that sends the last output message of UTM to the client.

**DEST=Mandatory operand!**

**destination-name**
A partner application identification of up to 8 bytes. This name must be specified in the `upicfile` as the symbolic destination name (see page 303).

**.DEFAULT**
A default server is used.

Mandatory operand!
MODE=RR / CV
   Determines which communication paradigm is used for the service:
   RR request-response paradigm, default value
   CV conversational paradigm

BUFFERS=(subtype-1,...,subtype-n)
   List of subtype names that can be sent to the service (type X_OCTET is allowed always). Each name can be up to 16 bytes long.

   A separate BUFFER statement, which defines the characteristics of the particular subtype, must be specified for each of the subtypes listed here (see below). The BUFFERS= operand is sensitive to position and must (if specified) be the last operand of the statement.
   If BUFFERS= is omitted, only a buffer of type X_OCTET should be sent to the service (no type check is performed).
**BUFFER statement**

A BUFFER statement defines a typed buffer. Buffers of the same name must be defined in the same way on both the client side and the server side. Multiple definitions are not checked. The first buffer entry is valid, while all others are ignored.

Buffers of type “X_OCTET” have no special features and therefore do not require definition. Typed buffers are defined with the following parameters:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operand</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER</td>
<td>subtype-name</td>
<td>maximum 16 bytes</td>
</tr>
<tr>
<td></td>
<td>[,REC=referenced-record-name]</td>
<td>default: subtype-name</td>
</tr>
<tr>
<td></td>
<td>[,TYPE=X_COMMON / X_C_TYPE]</td>
<td>default: xatmigen sets TYPE automatically</td>
</tr>
</tbody>
</table>

**subtype-name**

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

**REC=referenced-record-name**

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

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

**TYPE=**

Type of buffer; for further details on types see page 281.

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

In the generation run, xatmigen also creates two additional operands with the following meaning:

**LEN=length**

length of the data buffer

**SYNTAX=code**

syntax description of the data structure in code representation, as specified in the table on page 282.
6.5.2 xatmigen or xtgen32 tool

xtgen 32 creates a local configuration file (LCF) from a file containing the local configuration definition (LC definition file) and one or more files containing C or COBOL data structures (LC description files), see diagram below:

![Diagram](attachment:diagram.png)

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

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

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

Under Windows systems, success and error messages are written to the program window.

Success and error messages are written to stdout and stderr under Unix systems.

Under BS2000/OSD, success messages and error messages are written to SYSOUT and SYSLST.

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

- In Windows systems, xatmigen is called with
  
  ```
  xtgen 32 [.exe] parameter
  ```
  
  `xtgen 32.exe` is located in the `xatmipath\ex` directory.

  `xatmipath` is the directory in which XATMI is installed. The `C:\Program Files\xatmi32` directory is set as the default.

- Under Unix systems, xatmigen is called with

  xatmigen `parameter`

  xatmigen can be found in the `upicpath/xatmi/ex` directory.

  where `upicpath` stands for the directory in which the openUTM client was installed.

- Under BS2000/OSD, you start xatmigen with the following command:

  `/START-PROGRAM $userid.SYSPRG.UTM-CLIENT.060.XATMIGEN`

  Enter options:

  * parameter

  `$userid` is the ID under which the openUTM client is installed.

  When entering the command, you can, of course, use lowercase letters in place of uppercase letters.

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

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

- `upic`

  If specified, a file `xtupic.def` containing entries for the generation of the `upicfile` is created. The file is written to the current directory.

  If specified, `upic` must be the first parameter in `xatmigen`. If the parameter is omitted, no generation statements are created.

- `-d..lcdf-name`

  Name of the LC definition file; mandatory specification.
**-l...lcf-name**
Name of the local configuration file to be created. The name must comply with the conventions of the respective operating system. It is advisable to choose a name with a maximum of 8 characters and add the extension “.lcf”.

Any existing LCF of the same name is automatically overwritten.

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

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

The -i switch takes priority over the -c switch, if this is specified. If **xatmigen** is running in the background or in batch mode, the -i switch must not be specified.

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

The following can be specified for stringcode (see table on page 282):

- C  octet string
- C! octet string, terminated by ‘\0’
- T  T.61 string
- T! T.61 string, terminated by ‘\0’

If no specification is made, T! is used.

Individual characters are also interpreted as T.61 strings (**stringcode**= T!). Lowercase letters c and t are also valid.

**descript-file-1...descript-file-n**
List of files containing the include or COPY elements with the data structures of the typed buffers.
If the list is omitted, only the type X_OCTET is allowed.
6.5.3 Configuring the carrier system and UTM partners

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

– with the UPIC carrier system, align the UPIC configuration (upicfile) with the local configuration and the partner generation
– align the initialization parameters specified in tpinit with the UTM application generation

6.5.3.1 Configuring UPIC

A side information file (upicfile) must be created for the carrier system UPIC. See figure 21 below to see which entries you must make in the upicfile, and how these correspond to the local configuration file and KDCFILE of the UTM partner. Operation without TNS is assumed. For more information, please refer to section “Side information for standalone UTM applications” on page 324.

<table>
<thead>
<tr>
<th>SD destination-name utmappl ...</th>
<th>Line in upicfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX APPLNAME=utmappl (or BCAMAPPL=utmappl)</td>
<td>UTM partner</td>
</tr>
<tr>
<td>SVCU DEST=destination-name ... TAC=tac</td>
<td>Entry in LCF</td>
</tr>
</tbody>
</table>

An entry must begin with SD (Unix systems or Windows systems), as code conversion between ASCII and EBCDI is performed by XATMI. The destination-name must be identical in the LCF and upicfile.

utmappl is the name of the UTM application, as generated in the KDCDEF statements MAX APPLNAME or BCAMAPPL=. Address information, such as IP address and port number, must be specified in the upicfile.

The transaction code tac in the SVCU statement must be defined with a TAC statement in the UTM generation.

If you specify the “upic” parameter for xatmigen, a upicfile is created in which the individual lines need only be extended to include the partner parameter (using an editor). If you do not specify the “upic” parameter, you must create the entire upicfile yourself.

Figure 21: Conformance considerations when configuring server and client (operation without TNS)
6.5.3.2 Initialization parameters and UTM generation

An XATMI client is initialized using the `tpinit` function. Parameters for the user ID, password, and local application name are passed in the `TPCLTINIT` structure. These parameters must be aligned with the UTM generation as described below.

**User ID and password**

This security function can only be used with the UPIC carrier system.

```
username=user  passwd=password  cltname=...      TPCLTINIT
```

```
USER  user, PASS=password      UTM
```

Figure 22: Aligning the generation parameters

If access control is active under openUTM with the UPIC carrier system, `user` and if necessary `password` must be specified both in the `tpinit` call and in a USER statement of the UTM generation.
Local name

The diagram below shows the initialization procedure in a case where a local application name is defined in the `upicfile` (operation without TNS using RFC1006).

```
username=... passwd=... cltname=client

TPCLTINIT

LNclient upicclt

upicfile

*** UPIC-PARTNER
PTERM upicclt ,PTYPE=UPIC-R
*** OPENCPIC-PARTNER
OSI-CON...,TSEL=upicclt

U T M

*) with UPIC-R always via TNS
```

Figure 23: Initialization of a local application (operation without TNS)

If a local application name is generated in the `upicfile`, this name can be specified for `tpinit` (`client` in this example).

The associated application name must then be the same as the name specified in the PTERM statement or in OSI-CON TSEL=.

If no local application name is generated in the `upicfile`, the name defined on the UTM side in the PTERM statement or in OSI-CON TSEL= must be specified (`upicclt` in this example).
Example

The sample extract below covers all the relevant steps in local configuration, UPIC configuration, initialization, and KDCDEF generation.

1. Client

   Local configuration:

   SVCU ...
   .RSN=SERVICE1
   .TAC=TAC1
   .DEST=SATURNUS

   ...

   upicfile:

   SDSATURNUS utmserv1

   Initialization

   TPCLTINIT tpinfo;

   strcpy (tpinfo.cltname, "CLIENT1");
   strcpy (tpinfo.usrname, "UPICUSER");
   strcpy (tpinfo.passwd, "SECRET");
   tpinit (tpinfo);

2. Server

   Local configuration

   SVCP SERVICE1 ... (REQP also possible)
   .TAC=TAC1

   KDCDEF statements

   MAX APPLINAME=UTMSERV1

   or

   BCAMAPPL UTMSERV1 (in BS2000, also with parameter TPROT=ISO)

   LTERM UPICTERM

   PTERM TNSCLIENT, PTYPE=UPIC-R, PRONAM=DxxxSyyy (with UPIC remote conn.)
   PTERM CLIENT1, PTYPE=UPIC-L (with UPIC local conn.)

   TAC TAC1, PROGRAM=..., API=(XOPEN,XATMI)

   USER UPICUSER,PASS=SECRET
6.6 Running XATMI applications

6.6.1 Linking and starting an XATMI program

6.6.1.1 Linking an XATMI program under Windows systems

You are advised to compile the XATMI program using the option __STDC__ (ANSI). When you link an XATMI client application, the following libraries must be included:

1. All client modules with the main program
2. The XATMI client library xatmipath\SYS\XTCLT32.LIB
   
   xatmipath is the path name under which XATMI was installed.
3. The UPIC DLLs and the PCMX DLL must be available.
4. If you wish to run XATMI with UPIC-L under Windows, you must link the library libxtclt.lib into your application program.

6.6.1.2 Linking an XATMI program under Unix systems

When linking an XATMI client application, the following libraries must be included.

1. All client modules with the main program
2. XATMI client library and UPIC library (see below)
3. -lm (abbreviation for the “mathlib” on Unix systems)

Depending on whether UPIC-L or UPIC-R is used, the following static or dynamic XATMI and carrier-system libraries must be linked:

- UPIC local carrier system:
  a) utmpath/upicl/xatmi/sys/libxtclt.a or .../libxtclt.so
  b) utmpath/upicl/sys/libupicipc.a or .../libupicipcp.so

utmpath is the path name under which openUTM was installed.
– UPIC remote carrier system:
  a) `upicpath/xatmi/sys/libxtclt.a` or `.../libxtclt.so`
  b) CMX: `upicpath/sys/libupiccmx.a` or `.../libupiccmx.so`  
    Socket: `upicpath/sys/libupicsoc.a` or `.../libupicsoc.so`
  c) CMX library

where `upicpath` stands for the directory in which the openUTM client was installed.

### 6.6.1.3 Linking an XATMI program under BS2000/OSD

The following libraries must be linked in when you link an XATMI client application:

1. All client modules with a main program
2. The XATMI client and UPIC library `$userid.SYSLIB.UTM-CLIENT.060`

`$userid` is the ID under which UPIC-R is installed.

The library `$userid.SYSLIB.UTM-CLIENT.060` contains the example BIND-TPCALL for linking an XATMI program.

### 6.6.1.4 Starting the program

An XATMI client program is started as an executable program.
6.6.2 Setting Environment variables on Windows and Unix systems

For XATMI applications, openUTM-Client interprets a number of environment variables. The environment must be set before the application is started. For diagnostics while an application is running, traces can be activated.

Environment variables

The following environment variables are interpreted for an XATMI application:

- **XPATH** Path name for trace files. If this variable is not set, the trace files are written to the current directory (= directory from which the XATMI application was started).

- **XTLCF** Name of the local configuration file (LCF). The file name of the local configuration file must comply with the operating system conventions. If this variable is not set, a search is made under the name `xatmilcf` in the current directory.

- **XTPALCF** Defines the search path for additional descriptions of typed buffers. The buffer descriptions are read from local configuration files with the name `xatmilcf` or from the name specified in XTLCF. A search for all important XATMI generations (e.g. SVCU ...) is performed in the local configuration file specified using XTLCF. A search for local configuration files is performed in all the directories specified in XTPALCF and the typed buffer descriptions are gathered internally (If multiple buffers have the same name only the first buffer description is used).

The search path structure is exactly the same as in the default Unix system variable PATH: `(directory1:directory2: ...)`. If the specified search path has more than 1024 characters the path is cut! You can make up to 128 LCF entries.

- **XTSVRTR** Trace mode for the XATMI application. Possible specifications:
  - **E** (error): activates the error trace
  - **I** (interface): activates the interface trace for XATMI calls
  - **F** (full): activates the full XATMI trace as well as the trace for sub-layers.
Running XATMI applications

Setting environment variables on Windows systems

Under Windows systems, you can set environment variables using the Start/Settings/Control Panel and then select System/Advanced/Environment. You can then create or expand the environment variables here. Under Windows systems, these settings remain valid until you change them again.

Setting environment variables on Unix systems

Under Unix systems, environment variables are set using the following command:

```
SET variablename = value
```

The environment variables are valid for one shell only; other values may apply for applications in another shell.

6.6.3 Setting job variables under BS2000/OSD

Job variables can be set for an XATMI application. They are linked to the application using the following link names:

<table>
<thead>
<tr>
<th>Link Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTPATH</td>
<td>Link to job variable containing the prefix for the names of the trace files. If this link name is not assigned to any job variable, the names of the trace files will be constructed without any prefix.</td>
</tr>
<tr>
<td>XTLCF</td>
<td>Link to job variable containing the file name of the Local Configuration File (LCF). The name of the Local Configuration File must comply with the operating system conventions. The system searches for the file under the current user ID. If XTLCF is not assigned to any job variable, the system searches under the name XATMILCF under the current user ID.</td>
</tr>
</tbody>
</table>
If the software product JV is loaded as a subsystem, the job variables can be set as follows under BS2000/OSD, for instance:

1. Create job variable:
   ```
   CREATE-JV JV-NAME=FULLTR
   ```

2. Pass value to job variable:
   ```
   MODIFY -JV JV[-CONTENTS]=FULLTR, SET-VALUE='F'
   ```

3. Set task-specific job variable link:
   ```
   SET-JV-LINK LINK-NAME=XTSVRTR, JV-NAME=FULLTR
   ```

4. Show task-specific job variable link:
   ```
   SHOW-JV-LINK JV[-NAME]=FULLTR
   ```

5. Delete task-specific job variable link:
   ```
   REMOVE-JV-LINK LINK-NAME=XTSVRTR
   ```

The job variables are task-specific under BS2000/OSD. Different job variables can be assigned to a second application running under the same ID.
6.6.4 Trace

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

The maximum size of a trace file is 128 Kbytes. As soon as this size is reached, a second file is activated. If this has also reached the limit, the first file is written again. For a client, a trace file has the following name:

- Unix systems and Windows systems
  - `XTCpid.n`
- BS2000/OSD:
  - `prefix.XTCtsn.n`

- `prefix` The part of the name specified in the job variable referred to by the link name XTPATH (without terminating period).
- `XTC` identifies an XATMI client trace
- `tsn` ID of the client task, 4-digit
- `n` number of the generation: 1 or 2

The more recent trace can be identified by the time stamp.

Example
- XTC00341.1: client trace file number 1
- XTC00341.2: client trace file number 2
6.7 Notes on changing from openUTM-Client < V4.0

tpcall, tpacall and tpconnect calls

As of openUTM V4.0, when tpcall(), tpacall() or tpconnect is called a check is performed to
determine whether the called service is to be included in the global transaction.
In C, the flag parameter is evaluated and the system checks whether the TPNOTRAN flag
is set (in COBOL, the contents of the TPTRAN-FLAG field are evaluated).
In openUTM V3.4, TPNOTRAN automatically applied when these calls were used. As of
V4.0 this flag must be set explicitly if you want to ensure that the program runs in the same
way as under V3.4.

Connecting to the carrier system with tpinit

As of XATMI V4.0, the introduction of a default server has meant that the default setting for
the cltname parameter in the tpinit() call has changed from UPICO000 to eight blanks.
If the default UPICO000 is used, the configuration must be changed. You can:

– change an existing entry for
  LNUPIPO000 localname
  into LN.DEFAULT localname

– or supply the parameter cltname with the value UPICO000
  before calling tpinit()

– or enter
  LN.DEFAULT UPICO000
  in the upicfile

Modifications for calling the xatmigen tool

The default string code has changed since openUTM V4.0.
In previous versions, if no -c switch was specified, the default stringcode was T; as of
openUTM V4.0, the default value is T! if no -c switch is specified.
6.8 xatmigen messages

xatmigen messages have the form XGnn mesagetext... and are output to stderr under Unix systems or to the program window under Windows systems and to SYSLST under BS2000/Osd.

Under Unix systems and Windows systems, use the LANG environment variable to control whether you want German or English messages.

Under BS2000/Osd, you can assign the language code ‘D’ or ‘E’ to a task-specific job variable with the link name LANG in order to control whether messages are issued in English or German.

XG01 Generation of the local configuration files: &LCF / &DEF / &CODE

**Meaning**
Start message of Tool.

&LCF name of local configuration file created

&DEF name of generation fragment created

&CODE string code for character array

XG02 Generation terminated successfully

**Meaning**
The LCF was created; generation was terminated successfully.

XG03 Generation terminated successfully with warnings

**Meaning**
The LCF was created. Nevertheless, a warning is output because unnecessary files were specified, for example. However, this warning has no effect on the generation.

XG04 Generation terminated by error
No file created.

**Meaning**
The LCF was not created; the generation could not be performed. The cause can be determined from previous messages

XG05 &FTYPE file'&FNAME'

**Meaning**
This message specifies the file currently being edited, in the following form:

&FTYPE: "description" file contains data structures

"definition" file contains the LCF input

"LC" file contains the local configuration

&FNAME: File name
XG10  Call: &PARAM

**Meaning**
Syntax error when calling XATMIGEN:
PARAM: possible call parameters and switches

XG11  [Error] Cannot create &FTYPE file 'FNAME &REASON

**Meaning**
The &FNAME file of type &FTYPE cannot be created
&REASON contains a more precise explanation.
&FTYPE: GEN = generation fragment file (=generation statements)
LC = local configuration file

XG12  [Warning] File not found.

**Meaning**
The definition file or a description file was not found; perhaps the file does not exist.

XG13  [Warning] Too many &OBJECTS, Maximum: &MAXNUM

**Meaning**
Message indicating that too many objects were found.
&OBJECTS: subtypes
&MAXNUM: maximum number

XG14  [Error] Line &LINE: Syntaxerror, &helptext

**Meaning**
Syntax error in line &LINE of the LC definition file
&HELPTEXT: help text

XG15  [Error] Line &LINE: No record definition found for buffer &BUFF

**Meaning**
No associated record definition could be found for the buffer &BUFF in line &LINE.

XG16  [Error] Line &LINE: Basicitype error in buffer &BUFF

**Meaning**
The syntax description of the buffer &BUFF in line &LINE of the LCF contains an incorrect basic type (int, short, etc.).

XG17  [Error] Cannot open &FTYPE file 'FNAME'.
&REASON

**Meaning**
The &FNAME file of type &FTYPE cannot be opened.
&REASON contains a more detailed explanation.
&FTYPE: DEF (= LC definition file)
**XG18**  
[Error] &REASON

**Meaning**  
General error.  
&REASON contains a detailed reason for the error.

**XG19**  
[Message] Created new buffer: '&BUFF'

**Meaning**  
&BUFF: created buffer

**XG20**  
[Message] Service name '&SVC' truncated to 16 characters!

**Meaning**  
&SVC: service name.

**XG21**  
[Message] Line &LINE: unknown statement line '&HELPTEXT'

**Meaning**  
Message for the line &LINE in the LC definition file  
&HELPTEXT: help text (part of LC-line)

**XG22**  
[Message] Line &LINE: Default set MODE='&TEXT'

**Meaning**  
Message for the line &LINE in the LC definition file  
&TEXT: set default mode
7 Configuration

A client with the UPIC carrier system always uses UTM applications as servers in Windows systems, Unix systems or BS2000/OSD. The configuration of the UPIC carrier system must therefore be coordinated with the generation of the UTM partner application(s).

Figure 24: Configuration with and without side information file
7.1 Configuration without upicfile

For communication between UPIC and UTM, it is necessary for both the UPIC client and the UTM server to sign on to the local communication system with a name. UPIC signs itself on to the communication system with the local_name, UTM with the BCAMAPPL (application name). A communication relationship between the client and server is defined by UPIC addressing the UTM application under its BCAMAPPL. UTM receives the local name of the client in order to be able to authenticate the client (PTERM statement).

The client must use the name of the remote system for addressing if the communication system permits global communication. In this case, the complete address of the UTM partner consists of BCAMAPPL and the system name.

UPIC addresses the UTM application using the partner_LU_name. A partner_LU_name is designated as single-part if it only contains the address information about the local name of the UTM partner application. The two-part partner_LU_name is identified by the fact that it contains a dot ("."). The part to the left of the dot is the application name, the part to the right of the dot is the system name. The dot itself does not form part of the address.

The values for TSEL and HOSTNAME are derived from the partner_LU_name. The left part, up to the period (".") i.e. the application name, is assigned to TSEL. The part to the right of the period, i.e. the host name, is assigned to HOSTNAME.

Address components

- local_name

  The local_name is set with the Enable_UTM_UPIC call. A preset local_name is used if an empty local_name (8 blanks and/or length = 0) is passed with this call. The local_name is assigned the following default value:

  X/W
  - UPICL with UPIC-L
  - UPICR with UPIC-R

  It is overwritten with the call Specify_Local_Tsel.

upicfile comparison

The value of local_name can be overwritten using a upicfile. The upicfile is described in section “The side information file (upicfile)” on page 323.
Configuration without upicfile

- **partner\_LU\_name**
  
  Following the Initialize\_Conversation call, the partner\_LU\_name is assigned the following default value:

  - UTM with UPIC-L
  - UTM.local with UPIC-R

  It is overwritten with the Set\_Partner\_LU\_Name call.

  **upicfile comparison**

  The value of partner\_LU\_name can also be overwritten using a upicfile. The partner\_LU\_name in turn is addressed using the Symbolic Destination Name in the upicfile.
  
  The upicfile is described in section “The side information file (upicfile)” on page 323.

- **Symbolic Destination Name**

  The Symbolic Destination Name is precisely 8 characters in length and is passed in the Initialize\_Conversation call. An empty Symbolic Destination Name consists of precisely 8 blanks.

  An empty Symbolic Destination Name must be passed as the Symbolic Destination Name in the Initialize\_Conversation call.

  **upicfile comparison**

  When a upicfile is being used, an empty Symbolic Destination Name can be passed in the Initialize\_Conversation call.

  The upicfile is described in section “The side information file (upicfile)” on page 323.

### 7.1.1 UPIC-L configuration

UPIC-L uses the mechanisms of interprocess communication on Windows and Unix systems. In these communication systems, the local\_name and the partner\_LU\_name can be directly mapped to the addressing formats of the communication system. You must bear in mind that the partner\_LU\_name is only ever allowed to be specified as single-part, because the UPIC-L client and the UTM partner application always run on the same system as a result of the communication system used. The specification of a two-part partner\_LU\_name would also contain a system address. A two-part partner\_LU\_name is treated as an error because it can never be used.
7.1.2 UPIC-R configuration

UPIC-R uses transport systems for communication. In almost all practical situations, this involves TCP/IP with the protocol referred to as RFC1006. Transport systems have their own address regulations. The RFC1006 protocol is characterized by the fact that each transport system application signs itself on to the transport system with a name, referred to as the transport selector (T-SEL). The partners address one another using these names. RFC1006 is based on TCP/IP, so TCP/IP also requires the following addressing information:

- System name
- Port number

For BS2000, it has been agreed to use port number 102 wherever possible.

There is no general recommendation with respect to the port number under Unix systems and Windows systems. Port number 102 should, however, be used with care.

UPIC-R is configured using local_name and partner_LU_name, with the local_name being mapped on the local T-SEL. The application name from the two-part partner_LU_name is mapped on the remote T-SEL, the system name from the two-part partner_LU_name is the name of the system in the network. The partner_LU_name must be two-part, otherwise the described procedure does not work.

When mapping the local_name and the application name to the T-SEL, bear in mind that the character code of the T-SEL is not defined a priori. The two systems on which the server and client are running can use different character codes for representing the T-SEL (e.g. Windows systems uses an extended ASCII character code, BS2000/OSD the EBCDIC character code). Consequently, the format of the names must be defined. Three character formats are possible between UPIC and UTM: ASCII, EBCDIC and TRANSDATA. The TRANSDATA character set is a restricted subset of the EBCDIC character set. UPIC-R checks whether the character set used by local_name and/or the character set used by the application name can be converted into the TRANSDATA character set. The TRANSDATA character format is used if this is the case, otherwise the EBCDIC character format is used.

One port number each is assigned to both the local_name and the partner_LU_name. The two port numbers are not derived from the name, they are always set to the value 102 by default.

The local port number is assigned to the local_name. The default value can be overwritten. The local port number is a purely formal value which does not have any effect, and is only entered on grounds of compatibility. It should be disregarded in the configuration of UPIC-R.
The remote port number is assigned to `partner_LU_name`. In contrast to the local port number, there is a significant importance attached to the remote port number. This is because the UTM partner application is addressed using the remote port number. In the vast majority of practical cases, it is sufficient to use the default value 102. BCAM and CMX always support port 102 as the central access port for RFC1006. Although it is possible to select another port, this requires a significant amount of configuration work on the server side, for example BCMAP entries have to be created for the BS2000/OSD system. Such configurations require a certain level of experience and are not described here. As a rule, port 102 cannot be used if the UTM partner application is running on a system which uses PCMX to access the transport system. In this case, the value of the remote port number must be overwritten with the value which is used by the UTM application.

The values T-SEL, T-SEL format and local port number of the `local_name` can be overwritten with the following calls:

- `Specify_Local_Tsel`
- `Specify_Local_Tsel_Format`
- `Specify_Local_Port`

The values can also be overwritten by entries in the `upicfile`. In this case, the corresponding values are defined using keywords. The `upicfile` is described in section “The side information file (upicfile)” on page 323.

The addressing information for the network can be formed by specifying the `local_name` and using the internal rules of UPIC to have the network address created. It is also permitted and a function has been provided to overwrite one or more of the values derived from the `local_name` using the specified calls. It is permitted for any mixture of derived, default and explicitly set values to be used in this case. Equally, it is permitted for all of the values derived from the `local_name` to be overwritten. The `local_name` is meaningless if you select this type of configuration. You can then specify any `local_name` whatsoever, only providing it is compliant with the formal criteria of the `Enable_UTM_UPIC` call.

The values system name (or the Internet address derived from it), T-SEL, T-SEL format and remote port number can be overwritten with the following calls:

- `Set_Partner_Host_Name`
- `Set_Partner_IP_Address`
- `Set_Partner_Tsel`
- `Set_Partner_Tsel_Format`
- `Set_Partner_Port`

The `Set_Partner_Host_Name` call is ignored if the `Set_Partner_Host_Name` and `Set_Partner_IP_Address` calls are both called. The values can also be overwritten by entries in the `upicfile`. In this case, the corresponding values are defined using keywords. The `upicfile` is described in section “The side information file (upicfile)” on page 323.
In many cases, the addressing information for the network can be formed by specifying the partner_LU_name and using the internal rules of UPIC to have the network address created. It is also permitted and a function has been provided to overwrite one or more of the values derived from the partner_LU_name using the specified calls. It is permitted for any mixture of derived, default and explicitly set values to be used in this case. Equally, it is permitted for all of the values derived from the partner_LU_name to be overwritten. The partner_LU_name is meaningless if you select this type of configuration. You can then specify any partner_LU_name whatsoever, only providing it is compliant with the formal criteria which are required of it (among other aspects, it must be two-part).

### 7.1.3 Configuration with TNS entries

UPIC-R can also be configured using TNS entries if UPIC-R is using the transport system components CMX or PCMX for communication. UPIC-R with CMX always first tries to find a global name in the TNS directory for the local_name and the partner_LU_name. If a global name is found for the local_name and/or the partner_LU_name is found, then it is used. All other configuration settings are ignored. However, the configuration takes place as described above if no TNS entry is found for the local_name and/or the partner_LU_name.

### 7.1.4 Configuration using BCMAP entries

If UPIC uses the transport system component CMX(BS2000) for communication on BS2000/OSD, the configuration is influenced by BCMAP entries.

BCMAP entries for the client application and for the UTM partner application are only necessary in a few exceptional cases where communication takes place with a UTM application on Windows systems.

The UPIC client cannot influence the effect of BCMAP entries.

BCMAP entries can be created both for the local_name and for the partner_LU_name.

BCMAP entries for the local_name are not recommended.

BCMAP entries for the partner_LU_name are generally required if a UPIC client on BS2000/OSD is to communicate with a UTM application on Windows systems.
7.2 The side information file (upicfile)

You must create the upicfile yourself. This file has the following format:

- In Windows and Unix systems the file must contain only text and must be called upicfile. If you choose a different name, you must also set the UPICFILE environment variable accordingly.

- You must create a SAM file with the name upicfile in BS2000. If you choose a different file name, you must set the job variable UPICFIL accordingly.

This file is used by all client programs, e.g. in the Initialize_Conversation or Enable_UTM_UPIC calls.

The side information file is accessed with the environment variable or job variables UPICPATH. This allows you to specify the directory in which the file is located. If the variable is not set, the system looks for the file in the current directory.

The upicfile recognizes the following types of entries:

- communication partner entries which are addressed in the client program using the symbolic destination name.

- Side information entries for those communication partners in an openUTM-Cluster that are addressed using the symbolic destination name in the client program.

- Side information entries for the local application which are addressed in the client program using the local application name. These entries are optional.

To make the layout of the upicfile legible, the file may also contain blank lines and/or comment lines. Comment lines are identified by an asterisk ("*") in column 1. Note that a semicolon is always interpreted as an end-of-line character, even within a comment line.
7.2.1 Side information for standalone UTM applications

Each communication partner is addressed in the client program by its symbolic destination name. This name is specified when a conversation is initialized (in the `Initialize_Conversation` call).

An entry must be created in the `upicfile` for every `Symbolic Destination Name` which is used in the program. Each entry takes up one line in the `upicfile`.

The entry takes the following form for standalone UTM applications:

<table>
<thead>
<tr>
<th>SD or HD</th>
<th>symbolic destination name</th>
<th>blank</th>
<th>partner_LU_name</th>
<th>blank</th>
<th>transaction code</th>
<th>blank</th>
<th>keywords</th>
<th>end-of-line character</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>8 bytes</td>
<td>1 byte</td>
<td>1-32 bytes</td>
<td>1 byte</td>
<td>1-8 bytes</td>
<td>1 byte</td>
<td>optional</td>
<td>optional</td>
</tr>
</tbody>
</table>

1 With local connection via UPIC local, “partner_LU_name” can only be up to 8 bytes long.

**Description of the entry:**

- The names specified in the entry must be separated by blanks.
  Exception: There must be no blank between the identifiers SD/HD and the symbolic destination name.

- Identifiers SD/HD:
  The line begins with the identifier SD or HD. This specifies whether or not UPIC is to perform automatic code conversion during sending and receiving of data.
  For more information on code conversion, see also section “Code conversion” on page 104.

Windows and Unix systems:

- If HD is specified, automatic code conversion of the user data takes place during sending and receiving.
- Data which is sent to the UTM partner application is converted from the locally used code to EBCDIC.
- Data arriving from the partner application is converted from EBCDIC into the local code.
- If you specify SD, no automatic code conversion takes place.
BS2000/OSD:

The codes have the opposite meanings in BS2000/OSD.

In UPIC on BS2000/OSD, HD means that no automatic code conversion is carried out when sending and receiving data in the local system. HD should always be specified if the client communicates with the UTM application on BS2000/OSD (BS2000/OSD - BS2000/OSD link).

SD means that an EBCDIC->ASCII conversion will be carried out before sending data and an ASCII->EBCDIC conversion will be carried out on receiving data. SD should only be used for connections to UTM applications on Unix systems or Windows systems.

The SD/HD identifier in the upicfile can be overwritten with the `Set_Conversion` call.

- **symbolic destination name**
  The symbolic destination name must be precisely eight characters long.

- **partner_LU_name**
  With connections via UPIC remote, the `partner_LU_name` can be between 1 and 32 characters long. For `partner_LU_name` you must specify the symbolic name under which the UTM partner application is known to the communication system.
  With connections via UPIC remote you should always specify the `partner_LU_name` in two levels (separated by a period) in the format `applicationname.processorname`. The values for TSEL (=`applicationname`) and HOSTNAME (=`processorname`) are derived from the two-part `partner_LU_name`.

  You have to specify the `partner_LU_name` in two parts in BS2000. `processorname` must then match the name of the remote computer in BCAM-RDF.

  **Example:**

  **Specification in the upicfile:** SDsymbdest UTMAPPL1.D123ZE45

  An entry in the upicfile can be overwritten with the `Set_Partner_LU_Name` call.

  The individual values of a two-level `partner_LU_name` can be overwritten by entries in the side information file (HOSTNAME=, TSEL=) or by using the calls `Set_Partner_Hostname` and `Set_Partner_Tsel`.

  **UPIC-L:**

  With local connection to a UTM application via UPIC-L, the partner name must not exceed 8 characters and must be specified in one level.
transaction code (optional):
You can specify the transaction code of a UTM service. The transaction code is between 1 and 8 characters long. The transaction code you specify must have been generated in the UTM partner application (TAC statement) or dynamically configured. Specification of a transaction code in an entry is optional. If it is not specified, the transaction code (name of the service) in the program must be given in the Set_TP_Name call.

An entry in the upicfile can be overwritten with the Set_TP_Name call.

keywords (all entries are optional)
The following keywords can be used to influence the UPIC-specific conversation characteristics (see also section “CPI-C terms” on page 87) in the upicfile. The keywords are used to enter addressing information and to specify whether encryption is to be implemented.

You can enter keywords either after the partner name or after the transaction code. Keywords must be separated from the partner name or transaction code by a space. You can enter as many keywords as you like in any order. When entering more than one keyword, you must use a space to separate them.

ENCRIPTION-LEVEL={NONE | 0 | 1 | 2 | 3 | 4}
ENCRIPTION-LEVEL is used to specify whether or not the data for the conversation is to be encrypted and which encryption level is to be used.

If you enter ENCRYPTION-LEVEL=NONE or ENCRYPTION-LEVEL=0 (both have the same effect), the user data is not encrypted. If the UTM application establishes a connection which demands encryption of data then the encryption level is automatically adjusted accordingly. The same happens if UPIC on a connection with ENCRYPTION-LEVEL=NONE calls a TAC which is generated using encryption and UPIC does not send user data when calling the TAC. When UPIC receives encrypted data, the value of the encryption level is automatically increased accordingly.

If you specify ENCRYPTION-LEVEL=1, 2, 3 or 4 and openUTM can implement this encryption on the connection, all user data of the subsequent conversation is encrypted with the same level before transfer.
Values 1 to 4 mean:

1. The user data is encrypted using the DES algorithm. An RSA key with a key length of 200 bits is used for exchange of the DES key.

2. The user data is encrypted using the AES algorithm. An RSA key with a key length of 512 bits is used for exchange of the AES key.

3. The user data is encrypted using the AES algorithm. An RSA key with a key length of 1024 bits is used for exchange of the AES key.

4. The user data is encrypted using the AES algorithm. An RSA key with a key length of 2048 bits is used for exchange of the AES key.

The conversation is ended if openUTM does not support the specified encryption level.

The value is ignored if the UTM application cannot implement encryption for one of the following reasons:

– you have not installed openUTM-Crypt
– it does not want to implement encryption because the client partner was generated as 'trusted'.

The value of ENCRYPTION-LEVEL is ignored.

The entry in the upicfile can be overwritten using the Set_Conversation_Encryption_Level call.

HOSTNAME=

The host name is the processor name and can be up to 32 characters in length. The host name overwrites the value assigned using Initialize_Conversation.

An entry in the upicfile can be overwritten using the Set_Partner_Host_Name call.

The value of HOSTNAME is ignored.

IP-ADDRESS=nnn.nnn.nnn.nnn or =x: x: x: x: x: x: x: x (IPv6)

You can enter an Internet address in IPv4 or IPv6 format.

– If the Internet address is specified using traditional dot notation, it is interpreted as anIPv4 address.

– If the Internet address is specified in the form x: x: x: x: x: x: x: x, it is interpreted as an IPv6 address. x represents a hexadecimal number between 0 and FFFF. The alternative methods of writing IPv6 addresses (e.g. the omission of zeros using :: or IPv6 mapped format) are permitted.

If an Internet address is entered, the value of HOSTNAME is ignored.
An entry in the upicfile can be overwritten using the Set_Partner_IP_Address call.

UPIC-L  The value for IP-ADDRESS is ignored.

UPIC on BS2000/OSD using CMX as its communication system
The value for IP-ADDRESS is ignored.

PORT=listener-port
The port number is only entered for the address format RFC1006. The port number can be a value between 0 and 32767. The port number overwrites the port-number value assigned using Initialize_Conversation. Entering PORT is optional. If operation without TNS is specified for this communication partner, the value of PORT is used as the port number and not 102.

An entry in the upicfile can be overwritten using the Set_Partner_Port call.

UPIC-L  The value of PORT is ignored.

UPIC on BS2000/OSD using CMX as its communication system
The value for PORT is ignored.

PROTOCOL= {34 | 40}
In PROTOCOL you specify whether communication is to take place via the UPIC protocol V4.0 (PROTOCOL=40) or via the UPIC protocol V3.4 (PROTOCOL=34). Specification of PROTOCOL is optional.

PROTOCOL=34 is required for conversations with UTM applications < V4.0.

If you do not specify PROTOCOL, UPIC first attempts to establish a conversation based on the extended protocol (40). If this is unsuccessful, UPIC then tries to establish the conversation based on the V3.4 protocol (34). UPIC-R using the Socket communication system do not uses this automatism.

If the conversation is established based on the V3.4 protocol, the UPIC functions „Exchange format data“ and „Activate function keys“ cannot be used, for example. If you do specify PROTOCOL, UPIC attempts to establish the conversation based on the specified protocol only. If you specify PROTOCOL=40 and the attempt to establish the conversation based on the extended protocol fails because the UTM server does not support this version of the UPIC protocol, then UPIC does not try to establish the conversation based on V3.4 of the protocol.

A UTM server as of version 4.0 detects, when establishing the conversation, which protocol the client supports.

An entry in the upicfile can be overwritten with the Set_Communication_Protocol call.

UPIC-L  The value of PROTOCOL is ignored.
RSA-KEY=rsa-key
The public part of the RSA key of the partner application can be entered. If the public key is entered, the UPIC library compares the entered key with the one it received from the UTM partner application on connection setup. If there is a difference between keys, whether it be a change of at least one byte or just a change in length, the connection to is cleared down immediately by the UPIC library. This procedure is used to check whether the key is genuine.

UPIC-L The value of RSA-KEY is ignored.

T-SEL=transport-selector
The transport selector (T-SEL) of the transport address addresses the partner application within the remote system. It must be the same as the entry in the remote system. The transport selector is a name and can be up to 8 characters long. The specified T-SEL overwrites the value assigned using Initialize_Conversation. The use of T-SEL is optional.

The entry in the upicfile can be overwritten using the Set_Partner_Tsel call.

UPIC-L The value of T-SEL is ignored.

T-SEL-FORMAT={T | E | A}
TSEL-FORMAT is the format indicator of the transport selector. The valid formats are:

T for TRANSDATA
E for EBCDIC
A for ASCII

TSEL-FORMAT overwrites the value assigned using Initialize_Conversation. The use of T-SEL-FORMAT is optional.

If operation without TNS is specified for a communication partner, the value of TSEL-FORMAT is used. The entry in the upicfile can be overwritten using the Set_Partner_Tsel_Format call.

UPIC-L The value of T-SEL-FORMAT is ignored.
End-of-line character:
The character that concludes the entry varies depending on the platform for which the upicfile is created:

- **Windows systems:**
  Each line is concluded with a carriage return and line feed (the return key). A semicolon before the carriage return is optional.

- **Unix systems:**
  The line is concluded with a <newline> character (line feed). A semicolon before the <newline> character is optional.

- **BS2000/OSD:**
  The end of line is represented by a semicolon (;). No spaces are permitted after this.

If there is a semicolon in a line (contents of the side information entry), UPIC treats this as the end of the line and interprets the rest of the line as a new line (until the next end-of-line character).

Note that in BS2000/OSD, the next end of line character is also a semicolon. BS2000 editors such as EDT have a different view of lines from UPIC. If a further blank follows the semicolon of line \( n \) in the editor and line \( n+1 \) starts with SD and ends with a semicolon, UPIC sees a line which starts with " SD" and not with "SD".

The "Symbolic Destination Name" in this line is not found.
**Defining a DEFAULT server**

For your client application you can define a DEFAULT server or a DEFAULT service (see also section “Default server and DEFAULT name of a client” on page 130). A client program is connected to the DEFAULT server/service if in the program an empty name is passed as a symbolic destination name. In the DEFAULT entry you enter the value `.DEFAULT` instead of the symbolic destination name. The DEFAULT server entry must therefore have the following format:

<table>
<thead>
<tr>
<th>SD or HD</th>
<th>.DEFAULT</th>
<th>blank</th>
<th>partner LU name</th>
<th>blank</th>
<th>transaction code</th>
<th>blank</th>
<th>keywords</th>
<th>end-of-line character</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2 bytes</td>
<td>1 byte</td>
<td>1-32 bytes²</td>
<td>1 byte</td>
<td>1-8 bytes</td>
<td>1 byte</td>
<td></td>
</tr>
</tbody>
</table>

1 With a local connection via UPIC local, “partner_LU_name” can only be up to 8 bytes long.

With such an entry you define the UTM partner application `partner_LU_name` as the DEFAULT server. If you specify a transaction code, you also define the associated service as the DEFAULT service. You can call a different service on the DEFAULT server by setting a different transaction code in the program with the `Set_TP_Name` call (e.g. KDCDISP for the service restart). The specification in `Set_TP_Name` overwrites the value of `transactioncode` in the side information entry.
7.2.2 Side information for UTM cluster applications

Every communication partner, including UTM cluster applications is addressed by its symbolic destination name in the client program. This name is specified when a conversation is initialized (Initialize_Conversation call). You must make entries in the upicfile for each symbolic destination name used in the program.

A UTM cluster application is made up of several identical node applications running on the individual nodes of the cluster. To allow a UPIC client to easily access all the node applications of a UTM cluster application, you must configure an openUTM cluster in the upicfile. In doing this, you must observe the following rules.

Rules for configuring an openUTM cluster application

- For each symbolic destination name, you must create a separate entry for each node application in the upicfile with the code CD. If, for instance, the UTM cluster application is made up of three node applications, you must create three entries using the same symbolic destination name.

- All entries for a given symbolic destination name must follow each other consecutively. See the example on page page 337.

- The entries for a given symbolic destination name differ only in terms of the address specifications for the node (partner_LU_name or, if used, the keywords HOSTNAME and IP-ADDRESS). The specifications for transaction-code and the other keywords must match.

Format of an entry

Each entry occupies one line in the upicfile. An entry takes the following form:

<table>
<thead>
<tr>
<th>CD</th>
<th>symbolic destination name</th>
<th>blank</th>
<th>partner_LU_name</th>
<th>blank</th>
<th>transaction-code</th>
<th>blank</th>
<th>keywords</th>
<th>end of line character</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>8 bytes</td>
<td>1 byte</td>
<td>1-32 bytes</td>
<td>1 byte</td>
<td>1-8 bytes</td>
<td>1 byte</td>
<td>optional</td>
<td>optional</td>
</tr>
</tbody>
</table>
Description of the entry

- The names specified in the entry must be separated by blanks.
  Exception:
  No blank is permitted between the CD code and the symbolic destination name.

- CD code:
  The line starts with the code CD. This code has no effect on automatic code conversion
  (see also “CONVERSION={IMPLICIT | NO}” on page 336).

- symbolic destination name
  The symbolic destination name must be exactly 8 characters long.
  The combination CD-symbolic_destination_name can occur any number of times in the
  upicfile.

- partner_LU_name
  The partner_LU_name can be between 1 and 32 characters in length.
  The symbolic name under which the UTM partner application is known to the system
  must be specified for partner_LU_name.

  You should always specify partner_LU_name on two levels in the form
  applicationname.processorname (separated by a dot). The values for TSEL
  (=applicationname) and HOSTNAME (=processorname) are derived from the two-level
  partner_LU_name.

  In BS2000, you must specify the partner_LU_name on two levels. processorname must
  then match the name of the remote host in BCAM-RDF.

  Example
    Specification in the upicfile: CD-symbdest UTMAPPL1.D123ZE45

  An entry in the upicfile cannot be overwritten by a Set_Partner_LU_Name call. The
  individual values of a two-level partner_LU_name must not be overwritten in the program.
  Any such call will be rejected.

- transaction-code (optional specification):
  The transaction code of a UTM service can be specified. The transaction code is a
  name of up to 8 characters in length. The specified transaction code must have been
  generated in the UTM partner application (TAC statement) or must have been
  configured dynamically.

  Specification of a transaction code in an entry is optional. If this specification is omitted,
  the transaction code (name of the service) must be specified in the program with the
  Set_TP_Name call.

  An entry in the upicfile can be overwritten by a Set_TP_Name call.
Keywords (all specifications optional)
You can influence the UPIC-specific conversation characteristics (see also “Conversation characteristics” on page 87) in the upicfile with the following keywords. You use the keywords to specify the addressing information and specify whether encryption is to be used. You can specify the keywords after the partner name or after the transaction code, separated by blanks in each case. The sequence and number of keywords is arbitrary. Multiple keywords are separated by blanks.

ENCRYPTION-LEVEL={NONE | 0 | 1 | 2 | 3 | 4}
ENCRYPTION-LEVEL specifies whether the data for the conversation is to be encrypted or not and what encryption level is to be used.

If you specify ENCRYPTION-LEVEL=NONE or ENCRYPTION-LEVEL=0 (both have the same effect), the user data is not encrypted. If, however, the UTM application requires the data to be encrypted over a given connection, the encryption level is automatically increased. The same thing happens if UPIC calls a TAC generated with encryption over a connection with ENCRYPTION-LEVEL=NONE and UPIC does not send any user data when calling the TAC. If encrypted data is received, UPIC automatically increases the value for the encryption level.

If you specify ENCRYPTION-LEVEL=1, 2, 3 or 4, and openUTM is able to encrypt the data accordingly over the connection, all the user data of the following conversation is transmitted in encrypted form using the same level.

The values 1 through 4 have the following meanings:

1 Encryption of the user data using the DES algorithm. An RSA key with a key length of 200 bits is used to exchange the DES key.

2 Encryption of the user data using the AES algorithm. An RSA key with a key length of 512 bits is used to exchange the AES key.

3 Encryption of the user data using the AES algorithm. An RSA key with a key length of 1024 bits is used to exchange the AES key.

4 Encryption of the user data using the AES algorithm. An RSA key with a key length of 2048 bits is used to exchange the AES key.

If openUTM does not support the specified encryption level, the conversation is terminated.

The value is ignored if a UTM application cannot perform encryption because
– openUTM-Crypt is not installed
– it does not wish to perform encryption because the client partner has been generated as trusted
HOSTNAME=hostname

The hostname is the processor name and can be up to 32 characters in length. The hostname overwrites the value assigned with Initialize_Conversation.

An entry in the upicfile cannot be overwritten by a Set_Partner_Host_Name call.

IP-ADDRESS=nnn.nnn.nnn.nnn (IPv4) or = x: x: x: x: x: x: x: x (IPv6).

An Internet address can be specified in IPv4 and IPv6 format.

- If the Internet address is specified using the traditional dot notation, it is interpreted as an IPv4 address.

- If the Internet address is specified in the form x: x: x: x: x: x: x: x, it is interpreted as an IPv6 address. In this notation, x is a hexadecimal number between 0 and FFFF. The alternative notations for IPv6 addresses (e.g. the omission of zeros using :: or IPv6 mapped format) are permitted.

If an Internet address is specified, the value of HOSTNAME is ignored. An entry in the upicfile cannot be overwritten by a Set_Partner_IP_Address call.

UPIC on BS2000/OSD with CMX as the communication system

The value for IP-ADDRESS is ignored.

PORT=listener-port

The port number is only specified for the address format RFC1006. The port number can assume a value of 0 through 32767. This port number overwrites the value for the port number assigned with Initialize_Conversation. The PORT specification is optional.

If operation without TNS is specified for this communication partner, the value of PORT is used as the port number instead of 102.

An entry in the upicfile can be overwritten by a Set_Partner_Port call.

UPIC on BS2000/OSD with CMX as the communication system

The value of PORT is ignored.
The side information file (upicfile)

- **RSA-KEY=rsa-key**
  The public part of the RSA key of the partner application can be specified. If the public key is specified, the UPIC the library compares the specified key with the key it receives from the UTM partner application when the connection is established. If the two keys differ in at least one byte or even just in length, the connection is immediately cleared again by the UPIC library. This procedure allows the genuineness of the key to be checked.

  **T-SEL=transport-selector**
  The transport selector (T-SEL) of the transport address addresses the partner application within the remote system. It must match the specifications in the remote system. The transaction selector is a name of up to 8 characters in length. The T-SEL specified overwrites the value assigned with `Initialize_Conversation`. The T-SEL specification is optional.

  The entry in the `upicfile` can be overwritten by a `Set_Partner_Tsel` call.

  **T-SEL-FORMAT={T | E | A }**
  T-SEL-FORMAT is the format indicator of the transport selector. The valid formats are as follows:
  
  T  for TRANSDATA
  E  for EBCDIC
  A  for ASCII

  T-SEL-FORMAT overwrites the value assigned with `Initialize_Conversation`. The T-SEL-FORMAT specification is optional.

  If operation without TNS is specified for a communication partner, the value of TSEL-FORMAT is used. The entry in the `upicfile` can be overwritten by a `Set_Partner_Tsel_Format` call.

- **CONVERSION={IMPLICIT | NO}**
  CONVERSION=IMPLICIT specifies that automatic code conversion is performed on the user data on sending and receiving. For information on code conversion, see also the section “Code conversion” on page 104.

  If you do not specify CONVERSION= or if you specify CONVERSION=NO, no automatic conversion is performed.
End of line character:
The character used to terminate the entry differs for the various platforms for which the upicfile is created:

- **Windows systems:**
  Lines are terminated by a carriage return and line feed (Return key). A semicolon can be optionally inserted in front of the carriage return character.

- **Unix systems:**
  Lines are terminated with a <newline> character (linefeed). A semicolon can be optionally inserted in front of the <newline> character.

- **BS2000/OSD:**
  The end of the line is represented by a semicolon (;). No spaces are permitted after this.

If there is a semicolon in a line (contents of the side information entry), UPIC treats this as the end of the line and interprets the rest of the line as a new line (until the next end of line character).

Note that in BS2000/OSD, the next end of line character is also a semicolon. BS2000 editors such as EDT regard lines differently from UPIC.

If the semicolon in line \( n \) in the editor
- is followed by another blank and
- line \( n+1 \) starts with \( CD \) and ends with a semicolon,
  UPIC sees a line beginning with "CD" and not with "CD".
  The "symbolic destination name" in this line is not found.

**Example**

Two symbolic destination names (*service1* and *service2*) are to be configured for one UTM cluster application. The UTM cluster application is made up of three node applications on the hosts CLNODE01, CLNODE02 and CLNODE03. In addition, the upicfile contains a further entry for a standalone UTM application UMAPPL2.

The entries could, for instance, be as follows:

* entries for UTM cluster application UMAPPL1
  CDservice1 UMAPPL1.CLNODE01 TAC1
  CDservice1 UMAPPL1.CLNODE02 TAC1
  CDservice1 UMAPPL1.CLNODE03 TAC1

* entry for standalone application UMAPPL2
  SDservice2 UMAPPL2.D123S234 TAC4

The transaction code TAC1 can be overwritten in the program using *Set_TP_Name*, thus allowing other TACs to be addressed. In addition, it is possible to configure further standalone UTM applications (with the prefix SD or HD). These entries must, however, precede or follow the entries for the UTM cluster application described above.
Defining the DEFAULT server

You can define a DEFAULT server or a DEFAULT service for your client application (see also the section “Default server and DEFAULT name of a client” on page 130). A client program is connected to the DEFAULT server/service if an empty name is passed as the symbolic destination name in the program. In the DEFAULT entry, you specify the value .DEFAULT in place of the symbolic destination name. The DEFAULT server entry must therefore have the following format:

<table>
<thead>
<tr>
<th>CD</th>
<th>.DEFAULT</th>
<th>blank</th>
<th>partner_LU_name</th>
<th>blank</th>
<th>transaction-code</th>
<th>blank</th>
<th>keywords</th>
<th>end of line character</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 byte</td>
<td>1 byte</td>
<td>1-32 bytes</td>
<td>1 byte</td>
<td>1-8 bytes</td>
<td>1 byte</td>
<td></td>
<td>optional</td>
<td>optional</td>
</tr>
</tbody>
</table>

An entry such as this defines the UTM partner application partner_LU_name as the DEFAULT server. If you enter a transaction code, you also define the associated service as the DEFAULT service. You can call a different service on the DEFAULT server if you use the Set_TP_Name call in the program to set a different transaction code (e.g. KDCDISP for a service restart). The specification in Set_TP_Name overwrites the value of transaction-code in the side information entry.
### 7.2.3 Side information for the local application

For each client application several entries can be created in the **upicfile**. Each entry defines a local application name with which the client program can sign on to UPIC.

A side information entry for the local client application occupies one line and must have the following format:

```
LN    local application name    blank    application name    blank    keywords    end-of-line character
```

- **LN**: The line begins with the identifier LN. LN indicates that this is a side information entry for the local client application.
- **local application name**: Here you specify the local application name with which a client program signs on to UPIC. There must be no blank between the identifier LN and the local application name, but the local application name and the application name which follows it must be separated by a blank.
- **application name**: The application name can be up to 32 characters long. The client application signs on to the transport access system using the application name.

**UPIC local**: The application name can be up to 8 characters long.

---

1. With local connection via UPIC local, “application name” can only be up to 8 bytes long.
The side information file (upicfile)

- keywords (optional)
  The following keywords allow you to influence the UPIC-specific values for the local application (see also section “CPI-C terms” on page 87) in the upicfile. These keywords allow you to enter addressing information. Keywords can be entered after either the application name. You must separate the keyword by a space. You can enter as many keywords as you like and in any order. When entering more than one keyword, you must separate them with a space.

PORT=listener-port
  The port number is only entered for the address format RFC1006. The port number can be a value between 0 and 32767.

  If operation without TNS is specified for this communication partner, the value of PORT is used as port number instead of 102.

  An entry in the upicfile can be overwritten using the Set_Local_Port call.

  UPIC-L   The value of PORT is ignored.

T-SEL=transport-selector
  Is the transport selector (T-SEL) of the transport address. It must be the same as the entry in the remote system. The transport selector is a name which is up to 8 characters long. The use of T-SEL is optional.

  If operation without TNS is specified for a communication partner, the value of T-SEL is used. The entry in the upicfile can be overwritten using the Set_Local_Tsel call.

  UPIC-L   The value of T-SEL is ignored.

T-SEL-FORMAT={T | E | A}
  TSEL-FORMAT is the format indicator of the transport selector. The valid formats are:

  T   for TRANSDATA
  E   for EBCDIC
  A   for ASCII

  The use of T-SEL-FORMAT is optional.

  If operation without TNS is specified for a communication partner, the value of TSEL-FORMAT is used. The entry in the upicfile can be overwritten using the Specify_Local_Tsel_Format call.

  UPIC-L   The value of T-SEL-FORMAT is ignored.
● End-of-line character
  The end-of-line character depends on the platform:

  – *Windows systems:*
    Lines are terminated by a carriage return and line feed (Return key). A semicolon can be optionally used before the carriage return character.

  – *Unix systems:*
    The lines are terminated with the `<newline>` character (linefeed). A semicolon can be optionally used before the `<newline>` character.

  – *BS2000/OSD:*
    The end of line is represented by a semicolon (;). No spaces are permitted after this.

If there is a semicolon in a line (contents of the side information entry), UPIC treats this as the end of the line and interprets the rest of the line as a new line (until the next end-of-line character).

A local application name must always be specified for the local application in the `Enable_UTM_UPIC` call. If there is no entry in the `upicfile` for this local name or if the entry is invalid, the local name specified with `Enable_UTM_UPIC` is taken as the application name.

**Defining a default name**

In the `upicfile` you can define a DEFAULT name for your client application (see also section “Default server and DEFAULT name of a client” on page 130). The DEFAULT name is used whenever a client program passes an empty local application name at sign-on (`Enable_UTM_UPIC`). In the side information entry of the DEFAULT name you enter the value `.DEFAULT` instead of the local application name. The DEFAULT name entry must therefore have the following format:

<table>
<thead>
<tr>
<th>LN</th>
<th>.DEFAULT</th>
<th>blank</th>
<th>application name</th>
<th>blank</th>
<th>keywords</th>
<th>end-of-line character</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>1 byte</td>
<td>1-32 bytes</td>
<td>1 byte</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 With local connection via UPIC local, “application name” can only be up to 8 bytes long.
Whenever a client program passes an empty local application name at sign-on, UPIC uses this entry and signs the CPI-C program on to the transport access system with the application name specified in `application name`.

It is possible for several CPI-C programs to sign on to UPIC at the same time with the default name. These programs can even communicate with the same UTM application. But this is only possible if an LTERM pool with `CONNECT-MODE=MULTI` exists in the UTM application for connection of the client application (see also section “Multiple sign-on to the same UTM application with the same name” on page 131).
7.3 Coordination with the partner configuration

In Windows systems and Unix systems, the entries in the client program and the side information no longer necessarily have to be coordinated with the TNS entries in the TNS of the local system. If you are using UPIC-R without CMX (only with the Socket communication system), there is no need to use TNS entries and, indeed, it is not possible to do so. If you are using UPIC-R with CMX, you can create suitable TNS entries; if there are already suitable TNS entries for the local_name and/or for the partner_LU_name in the database, then these TNS entries are used first of all. This means the Specify_Local_Xxx() or Set_Parter_Xxx() calls and the keywords of the side information HOSTNAME, IP-ADDRESS, PORT, TSEL and TSEL-FORMAT have no effect.

If the client program is running under BS2000/OSD, BCMAP entries may be required, see also page 322.

There are dependencies between the entries in the client program, in the upicfile and the UTM generation. The following sections describe which parameters you must coordinate for partner configuration.

You can specify the information necessary for the transport system either using keywords directly in the upicfile or using function calls in the client program. If you do not use either of these options, the preset values will be used. The table below gives an overview of the preset values which can be modified in the side information file or in the program:

<table>
<thead>
<tr>
<th>Property</th>
<th>Function</th>
<th>Keyword</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>local application name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-SEL</td>
<td>Specify_Local_Tsel</td>
<td>T-SEL=</td>
<td>local application name</td>
</tr>
<tr>
<td>T-TSEL format</td>
<td>Specify_Local_Tsel_Format</td>
<td>T-SEL-FORMAT=</td>
<td>T</td>
</tr>
<tr>
<td>Port number</td>
<td>Specify_Local_Port</td>
<td>PORT=</td>
<td>102</td>
</tr>
</tbody>
</table>

| transport address         |                        |             |                              |
| T-SEL                     | Set_Parter_Tsel        | T-SEL=      | partner name                 |
| T-TSEL format             | Set_Parter_Tsel_Format | T-SEL-FORMAT= | T                           |
| Port number               | Set_Parter_Port        | PORT=       | 102                          |
| Internet address¹         | Set_Parter_IP_Address   | IP-ADDRESS= | Information from host        |
| Host name                 | Set_Parter_Host_Name   | HOSTNAME=   | Processor name               |

Table 14: Properties of the address information

¹ The Internet address takes priority over the host name.

The following relationships exist between the entries in the client program or in the upicfile and the generation of the UTM application.
Local application name

The local application name is specified in the calls `Enable_UTM_UPIC` and `Disable_UTM_UPIC`. A distinction is made between the following cases:

- The local application name is entered in the `upicfile` (identifier LN). The application name in this entry is transferred directly to the transport system.
- If the local application name is not entered in the `upicfile`, it is transferred as the application name directly by UPIC to the transport system.

**Partners on Unix systems or Windows systems or on BS2000/OSD without a BCMAP entry**

If the partner is a UTM application on a Unix system or Windows system or a UTM application on a BS2000/OSD for which no BCMAP entries have been generated, the generations must be coordinated as follows:

```
UPIC
local-name ptermname;

openUTM
PTERM ptermname
```

Both PTERM names must match. If there is no PTERM name generated for the client, there must be an LTERM pool via which the client can sign on.

**Partners on BS2000/OSD with a BCMAP entry**

If the partner is a UTM application on BS2000/OSD that uses BCMAP entries, the generations must be harmonized as follows:

```
UPIC
local-name tselname;

openUTM
PTERM ptermname
```

The T-selector of the local application must match the T-selector which is assigned to the client application in the server system.
Partner name

If the *partner_LU_name* (page 325) is specified in two parts (*tselname.processorname*), UPIC transfers this name directly to the transport system.

**Partners on Unix systems or Windows systems or on BS2000/OSD without a BCMAP entry**

If the partner is a UTM application on a Unix system or a Windows system or a UTM application on BS2000/OSD for which no BCMAP entries have been generated, the generations must be harmonized as follows:

```
UPIC
partner_LU_name utmsampl.processorname;
openUTM
BCAMAPPL utmsampl
```

The *applicationname* which UPIC transfers to the transport system must match the BCAMAPPL name of the UTM application via which the connection to the client is made (in the diagram this is *utmsampl*). *processorname* must be entered in the TCP/IP name service as the name of the remote system.

**Partners on BS2000/OSD with a BCMAP entry**

If the partner is a UTM application on BS2000/OSD that uses BCMAP entries, the generations must be harmonized as follows:

```
UPIC
partner_LU_name tselname.processorname;
openUTM
BCAMAPPL utmsampl
```

```
BCMAP
(tselname utmsampl)
```

*tselname* must match the T-selector of the BCMAP entry for the UTM application on the remote processor.
Coordination with the partner configuration
8 Implementing CPI-C applications

This chapter tells you what you need to know before and during implementation of CPI-C applications and what to do in the event of an error.

8.1 Runtime environment, linking, starting

Execution of CPI-C programs is controlled by environment variables or by the link name of the job variables in BS2000/OSD. The following tables list the variables necessary for this:

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPICPATH</td>
<td>Specifies the directory in which the side information file (\text{upicfile}) is stored. If the variable is not set, the file is sought in the current directory.</td>
</tr>
<tr>
<td>UPICFILE</td>
<td>Specifies the name of the side information file. If the variable is not set, the file name (\text{upicfile}) is set.</td>
</tr>
<tr>
<td>UPICLOG</td>
<td>Specifies the directory in which the log file is stored. The value that is assumed if the variable is not set depends on the platform used (see section “UPIC log file” on page 363).</td>
</tr>
<tr>
<td>UPICTRACE</td>
<td>Controls the creation of a trace, see page 364.</td>
</tr>
</tbody>
</table>
The following pages describe what you have to take into account when creating and implementing a CPI-C application on your system, depending on the platform used.

<table>
<thead>
<tr>
<th>Link name of the job variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPICPAT</td>
<td>Specifies the partially qualified file name [:catid:$userid.&lt;partial-name&gt;)] under which the side information file (upicfile) is stored. If the variable is not set, the system searches for the file under $userid.</td>
</tr>
<tr>
<td>UPICFIL</td>
<td>Specifies the right-hand part of the name of the side information file. If this variable is not set, the file name is set to upicfile. The complete file name is composed of UPICPAT.UPICFIL. If neither UPICPAT nor UPICFIL is set, the file name is &quot;$userid.UPICFILE&quot;.</td>
</tr>
<tr>
<td>UPICLOG</td>
<td>Specifies the partially qualified file name under which the logging file is to be stored. The value which is assumed if the variable is not set depends on the platform used (see the section &quot;UPIC logging file&quot;).</td>
</tr>
<tr>
<td>UPICTRA</td>
<td>Controls generation of a trace, see section “UPIC trace” on page 364.</td>
</tr>
</tbody>
</table>
8.1.1 Implementing in Windows systems

When creating and implementing CPI-C applications, you must take into account the special features described in section “Compilation, linking, starting” below and in section “Runtime environment, environment variables” on page 350.

When creating and implementing UPIC-local applications on Windows systems, you must also take into account the specifications described in section “Special features of implementing UPIC local on Windows systems” on page 351.

8.1.1.1 Compilation, linking, starting

When compiling and linking CPI-C applications on Windows systems, you must observe the following:

- Every CPI-C program requires the following header files for compilation:
  
  ```
  #include <WINDOWS.H>
  #include <upic.h>
  ```

  The header file `upic.h` is located in the directory `upic-dir\include`. `upic-dir` is the installation directory of openUTM-Client with the UPIC carrier system.

  This order of includes shown above is mandatory. It is advisable to compile the program using the option `__STDC__` (ANSI).

- When compiling CPI-C programs (UPIC remote only) you must set the following compiler options: `UTM_ON_WIN32`

  You can see the effect of this option in the header file `upic.h`. It is located in the directory `upic-dir\include`. `upic-dir` is the installation directory of the openUTM client with the UPIC carrier system.

- A CPI-C program consists of a series of modules which have to be linked to form a program. The following object modules are required for linking:
  
  - main program of the user
  - user modules
  - For programs which want to use PCMX:
    - the library `upicw32.lib`, located in the `upic-dir\SYS` directory.
  - For programs which do not want to use Socket interface:
    - the library `upicws32.lib` (socket), located in the `upic-dir\SYS` directory.

- Once the runtime environment has been made available (see below), you can start a CPI-C program just like any other program in Windows systems.
8.1.1.2 Runtime environment, environment variables

The environment variables listed in the table on page 347 are used for controlling CPI-C applications.

The path name can be given with blanks in the UPICTRACE variable. If blanks are used, then the path name must be enclosed in double quotes. Double quotes can also be used if there are no blanks in the path name.

Under Windows systems you set environment variables as follows:

– Select Start/Settings/Control Panel/System, select the Advanced tab in the System Properties dialog box and click on Environment Variables.

– Create/supplement the environment variables in the Environment Variables dialog box. These settings remain valid under Windows system until they are changed.

Proceed in the same way for other Windows systems. However, the menu commands and names may differ slightly.

There are user variables that apply only for the current user ID, and there are system variables that apply for all users. You must set system variables if you want to run a UPIC application as a service (a service runs without a user environment).

All these environment variables can also be set using the UPIC.INI file.

UPIC.INI

The environment variables for controlling a CPI-C application can be set with the help of the UPIC.INI file. Controlling UPIC via the UPIC.INI file has the advantage that environment variables can be set without Windows having to be closed down. If the UPIC.INI is used, it must be in the Windows directory and have the following structure:

```
[UPICW32DLL]
UPICPATH=directory
UPICTRACE=option
UPICLOG=directory
UPICFILE=name-side-information-file
```

Example:

```
[UPICW32DLL]
UPICPATH=C:\UPIC
UPICTRACE=-SX -dC:
UPICLOG=C:\UPIC\TMP
UPICFILE=upicfile
```

The entries in UPIC.INI are only read if the relevant environment variable has not been set.
It is no longer recommended that the file `UPIC.INI` is used. Since this file must be located in the Windows directory, not all users have unrestricted access to it. For instance, users who belong to the group `User` do not have write access to the file.

**Entries in the Registry**

The UPIC library supports the "IniFileMapping" mechanism. You can find further information on this in the "MSDN Library Visual Studio 6.0 - Platform SDK - Windows Base Services" under "WritePrivateProfileString()", for instance. The relevant key is `HKCU\Software\FSC\UPIC\UPICW32DLL`.

You can set up the values `UPICPATH`, `UPICTRACE`, `UPICLOG` and `UPICFILE` under the subkey `UPICW32DLL` (which corresponds to the "Section" entry in `UPIC.INI`) and enter the values described above in the data field.

That Registry values are only evaluated if the corresponding environment variable has not been set. They are, however, evaluated before the entries in `UPIC.INI`.

**CPI-C program resources**

- One file descriptor is reserved permanently for the trace file.
- If information is written to the log file, a file descriptor is used only during the write operation.
- Reading from the `upicfile` only requires a file descriptor during the `Enable_UTM_UPIC` call.
- Other resources are also used by the transport system.

**Special features of implementing UPIC local on Windows systems**

When implementing UPIC-local applications on Windows, you must bear in mind the special features described below.

**Linking UPIC-local applications**

When linking UPIC-local applications on Windows systems the following libraries are supplied:

- `utmpath\upic\sys\libupic\lib`, which must be linked to every client program (instead of `upicw32.lib`) and
- `utmpath\xatmi\sys\lib\libxtclt.lib`, which must also be linked to XATMI programs.

For further information on `utmpath`, refer to openUTM manual “Using openUTM Applications under Unix Systems and Windows Systems”.
Runtime environment

Executing the UPIC-local clients requires the dynamic libraries `utmpath\ex\libupicl.dll` and `utmpath\ex\libxtclt.dll`. These DLLs can be found via the environment variable PATH. PATH is extended accordingly when openUTM is installed.

Configuring a UPIC-local client with Visual C++

The following briefly describes how you can configure a UPIC-local client project using the Visual C++ Developer Studio. UPIC local is a component of openUTM for Windows systems and requires at least version 5 of the Visual C++ Developer Studio.

Client projects supplied with the openUTM Quickstart Kit are configured as described here.

To configure the project, select the Settings... command from the Project menu of the Visual C++ Developer Studio. The Project Settings dialog box is displayed on the screen. Now proceed as follows:

1. Link in the UPIC-local libraries `libupicl.lib` and `libxtclt.lib`:

   Select the Link tab sheet and make sure that in the Settings For list box the item All Configurations are marked.

   In the Category list box set the category to General, enter the name you want for the output file (`upicl.exe` here) and add the following libraries in the Object/Library Modules input field:

   - `libupicl.lib` for configuring CPI-C clients
   - `libxtclt.lib` and `libupicl.lib` for configuring XATMI clients (paying attention to the order: `libxtclt.lib` must come before `libupicl.lib`). A space must always be entered as the delimiter.

   These libraries must be entered in front of all existing `.lib` files. `utm-dir` stands for the installation directory of openUTM. If you enter search paths in Extras/Options in Developer Studio, you need not type in the full pathname here.
2. Configure debugger information:

Select the Link tab sheet and in the Settings For list box mark Win32Debug in the Settings For list.

In the Category list box, set the category to Debug and in Debug Info and select the Debug Info and Both Formats options in Debug Info.

3. Confirm your settings in Project Settings by clicking on OK.
8.1.2 Implementation in Unix systems

When creating and implementing CPI-C applications, you must take into account the special features described in section “Compilation, linking, starting” on page 354 and section “Runtime environment, environment variables” on page 355.

When creating and implementing UPIC-local applications in Unix systems, you must also take into account the specifications described in section “Special features of implementing UPIC local on Unix systems” on page 356.

8.1.2.1 Compilation, linking, starting

When compiling and linking CPI-C applications on Unix systems, you must observe the following:

- Every CPI-C program requires the following header file for compilation:
  ```c
  #include <UPIC.H>
  ```
  The header file is located in the `include` subdirectory of the UPIC installation directory.

- A CPI-C program consists of a set of modules which must be linked as a program using the C compiler of your system. The following object modules are essential for linking:
  - main program of the user
  - user modules

  For programs which use CMX/PCMX:
  - the system libraries `nsl.so, dl.so, socket.so` (not on every system) and `cmx.so`.
  - The library `cmx.so` must be linked in before the library `nls.so`.
  - the library `libupiccmx.a` (statically) or the library `libupiccmx.so` (dynamically), which can be found in the `upic-dir/sys/` directory.

  For programs which do not use CMX/PCMX:
  - the system libraries `nsl.so and dl.so`. On a few systems `socket.so` also
  - the library `libupicsoc.a` (statically) or the library `libupicsoc.so` (dynamically), which can be found in the `upic-dir/sys/` directory.

  For programs which do not use CMX/PCMX and want to use threading:
  - the system libraries `nsl.so, dl.so` and `socket.so`
  - the library `libupicsocmt.a` (statically) or the library `libupicsocmt.so` (dynamically), which can be found in the `upic-dir/sys/` directory.

An example showing all necessary library and link options can be found in the makefile for the sample program `uptac.c` in the `upic-dir/sample` directory.

- A CPI-C program is started just like any other program in Unix systems by entering the program name (note that the UTM application must be started beforehand).
8.1.2.2 Runtime environment, environment variables

The environment variables listed in the table on page 347 must be set in order to operate CPI-C applications:

- UPICPATH=directory
- UPICTRACE=option
- UPICLOG=directory
- UPICFILE=name-side-information-file
- export UPICPATH UPICTRACE UPICLOG UPICFILE

Resources of a CPI-C program

- A file descriptor is always required for the trace file.
- If data is written to the log file, a file descriptor is only required while the data is being written.
- To read from the upicfile, a file descriptor is only required during the Enable_UTM_UPIC call.
- Transport system resources are also required.

Signals

Signal handling routines can only be written in a CPI-C program for the signals SIGHUP, SIGINT and SIGQUIT. The CPI-C library functions are not interrupted by these three signals. This signal handling does not become effective until the current CPI-C function has terminated.

All other signals are prohibited!
8.1.2.3 Special features of implementing UPIC local on Unix systems

When implementing UPIC-local applications on Unix systems, you must also bear in mind the special features described below.

Linking UPIC local applications in Unix systems

When a CPI-C client application is connected locally to a UTM application on a Unix system, you must link in the library `libupicipc.a (static)` or `libupicipc.so (dynamic)` instead of `libupiccmx.a` or `libupiccmx.so`.

On Linux systems, the `–lcrypt` option must also be specified.

Environment variables

For controlling a UPIC-local application, the environment variable UTMPATH is also interpreted. UTMPATH must contain the name of the directory in which openUTM is installed.

Resources

With local connection, “shared memory” is used for communication with the UTM application. Access is via “shared memory keys” and is serialized with the aid of a semaphore. An additional file descriptor is reserved for shared memory.
8.1.3 **Deployment under BS2000/OSD**

You should take note of the special considerations listed below when deploying CPI-C applications in BS2000/OSD.

**Compilation, linking, starting**

The following applies when compiling and linking CPI-C applications on BS2000/OSD systems:

- Every CPI-C program requires the following include file in order to allow compilation:

  ```
  #include <UPIC.H>
  ```

  The include file is located in the library `$userid.SYSLIB.UTM-CLIENT.060`.

  `$userid` is the ID under which the openUTM client is installed.

- A CPI-C program comprises a set of modules which must be linked to form a single program. The following objects are required for linking:
  - main program of the user
  - User modules
  - For programs that wish to use CMX:
    - The system libraries `$sysid.SYSLNK.CRTE` and `$sysid.SYSLIB.CMX.013`
    - The libraries `$userid.SYSLIB.UTM-CLIENT.060.WCMX` and `$userid.SYSLIB.UTM-CLIENT.060`
  - For programs that wish to use Sockets:
    - The system library `$sysid.SYSLNK.CRTE`
    - The library `$userid.SYSLIB.UTM-CLIENT.060`
    - The appropriate Socket libraries

- You start a CPI-C in BS2000/OSD in the same way as any other program using the command `START-EXECUTABLE-PROGRAM`. 
Runtime environment

Execution of CPI-C applications under BS2000/OSD is controlled by the job variables. The link names of the job variables are listed in the table on page 348. You can set these as follows, for example:

/SET-JV-LINK LINK-NAME=*UPICPAT,JV-NAME=UPICPATH
/SET-JV-LINK LINK-NAME=*UPICFILE,JV-NAME=UPICFILE
/SET-JV-LINK LINK-NAME=*UPICLOG,JV-NAME=UPICLOG
/SET-JV-LINK LINK-NAME=*UPICTRACE,JV-NAME=UPICTRACE

Example:

/SET-JV-LINK LINK-NAME=*UPICTRACE,JV-NAME=UPICTRACE
/SET-JV-LINK LINK-NAME=*UPICPATH,JV-NAME=UPICPATH
/SET-JV-LINK LINK-NAME=*UPICFILE,JV-NAME=UPICFILE
/SET-JV-LINK LINK-NAME=*UPICLOG,JV-NAME=UPICLOG

Note that the link name assignment established with SET-JV-LINK is lost after LOGOFF. SET-VALUE='-r 128' controls the trace (see section “UPIC trace” on page 364).

8.2 Handling of CPI-C partners by openUTM

With a connection to a UTM application via CPI-C, some UTM functions cannot be used and some are used differently.

This relates to the following functions:

– INPUT exit and event service BADTAC
  With input from the CPI-C client, openUTM does not call the input exit or BADTAC.

– FPUT
  It is not possible to send an asynchronous message to a CPI-C client using FPUT. The KDCS call supplies the return code 44Z.

– PEND RS
  Under certain circumstances, PEND RS is handled like PEND FR for a CPI-C client; for further details, see the openUTM manual "Programming Applications with KDCS".
8.3 Behavior in the event of errors

This section describes the effects on a communication partner when a UTM server application or a CPI-C client application terminates. It also explains how to re-establish a basic state for successful program-to-program communication in the event of an error.

Termination of a UTM application

If the UTM application terminates, this is detected by the CPI-C program with the next call at the communication interface. The following two cases can be distinguished:

- a connection shutdown may be detected with a Receive call or
- the termination of the application may be detected with a call at the communication interface, which also caused the conversation to terminate automatically.

In both cases, CM_DEALLOCATED_ABEND is returned as the result.

Abnormal termination of a CPI-C program

The UTM application is generally informed of the program termination by means of a connection shutdown. In this case, no further actions are required.

If the UTM application does not detect a connection shutdown, the connection still exists as far as openUTM is concerned. Two cases can be distinguished:

- On the UTM side a PTERM or an LTERM pool with TPOOL ..., CONNECT-MODE=SINGLE is generated for the client application. In this case, openUTM can distinguish between the connected clients. As soon as a client attempts (after a loss of connection) to open another connection under the same name, openUTM shuts down the old connection and rejects the connection setup request. Any subsequent connection setup request from the client is then accepted.

- On the UTM side an LTERM pool with TPOOL ..., CONNECT-MODE=MULTI is generated for the client application. In this case, several clients can sign on to the UTM application from the same system and with the same name. The UTM application can then no longer recognize whether a client is signing on from scratch or after loss of a connection. A lost connection for which the UTM application was not shown a connection shutdown must in this case be shut down explicitly by the administration, i.e. openUTM does not shut down the “lost” connection itself the next time the client attempts to set up a connection.
**Behavior in the event of errors**

Implementing CPI-C applications

---

**Upic local:**

The following can occur:

The UTM application has not recognized the termination of the CPI-C process. As soon as the CPI-C program signs on to openUTM again with the same program name, openUTM shuts down the old connection and accepts the new one.

**Serious error in the CPI-C program**

If a serious error occurs while the UPIC program is running, and this error effectively prevents the program from continuing, the process is abnormally terminated (with FatalAppExit in Windows systems; with abort in Unix systems). The following error message is also written to the UPIC log file:

**UPIC: internal error <reason>**

The error messages that may occur on the CPI-C side are described in the table below.

<table>
<thead>
<tr>
<th>&lt;reason&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>When sending the rest of the data, the value of data length is negative</td>
</tr>
<tr>
<td>9</td>
<td>The SIGTRAP signal has occurred</td>
</tr>
<tr>
<td>10</td>
<td>Error when establishing the connection</td>
</tr>
<tr>
<td>11</td>
<td>Error when receiving confirmation for connection setup</td>
</tr>
<tr>
<td>12</td>
<td>Message other than connection setup received</td>
</tr>
<tr>
<td>13</td>
<td>Error when sending data</td>
</tr>
<tr>
<td>14</td>
<td>Error when receiving data</td>
</tr>
<tr>
<td>15</td>
<td>Invalid message received</td>
</tr>
<tr>
<td>16</td>
<td>Error when shutting down connection</td>
</tr>
</tbody>
</table>

For error diagnosis see also section “Diagnostics” on page 363.

---

**UPIC local:**

With local communication via UPIC local, moreover, error messages beginning with the letters “IPC” can occur. These come from openUTM and are described in the openUTM manual “Messages, Debugging and Diagnostics in Unix Systems and Windows Systems” under the dump error codes.

For error diagnosis you require the dump (e.g. core dump in Unix systems) together with the linked program as well as the contents of the UPIC trace file and the UPIC log file.
**Message exchange with a programmed PEND ER/FR**

If a programmed PEND ER/FR was carried out while a UTM program unit was running, the message segments sent with MPUT prior to the PEND ER/FR can be received. The `Receive` or `Receive_Mapped_Data` call is used for this purpose (until the return code is `CM_DEALLOCATED_ABEND`).

**Message exchange with SYSTEM PEND ER**

If, in the event of an error, the UTM service ends with PEND ER, the result `CM_DEALLOCATED_ABEND` is returned when `Receive` or `Receive_Mapped_Data` is called. In addition, an error message is written to the log file (see also section “UPIC log file” on page 363).

A separate error message for a UPIC-Client can be generated in a dialog program unit using the `MPUT ES` (error system) call (see also openUTM manual "Programming Applications with KDCS", `MPUT ES` call), which the UPIC client can read with he call `Receive` or `Receive_Mapped_Data`. In this case, no error message is written to the log file.

**Problems with connection setup**

Problems in setting up a connection to the UTM application can be detected by the fact that the `Allocate` call does not terminate with the result `CM_OK`. In this case you should check the following:

- **Use a ping command to check whether it is possible at all to establish a network connection between the client and server.**

  Call the `ping` command using:

  \[
  \text{ping <internetaddress> or ping <hostname>}
  \]

  `ping` must be in your path, i.e. the PATH variable must be suitably set.

  Under BS2000/OSD, call ping as follows:

  \[
  \text;/START-EXECUTABLE-PROGRAM -}
  \text{FROM-FILE=*LIBRARY-ELEMENT -}
  \text{(LIBRARY=$.SYSPRG.BCAM.XXX,ELEMENT-OR-SYMBOL=PING)}
  \]

- **Check the TCP/IP protocol using one of the standard applications `telnet` or `ftp`.**

  Call these commands as follows:

  \[
  \text{telnet internetaddress or telnet hostname}
  \]

  \[
  \text{ftp internetaddress or ftp hostname}
  \]

  The applications must be in your path, i.e. the PATH variable must be suitably set.
Under BS2000/OSD, the applications are called with:

- START-TELNET
- START-FTP

- Check whether the necessary resources are available in the UTM partner application. For example, the LTERM pool or the LTERM partner via which the client wants to sign on must not be locked. See also the openUTM manual "Generating Applications".

- Check whether all the necessary resources are available on the local system. You should always check the local generation (side information and TNS if necessary) and the partner generation (openUTM and TNS if necessary).

In a configuration which requires BCMAP entries in BS2000/OSD, you must make sure that the BCMAP command does not perform any update function, i.e. that BCMAP entries must first be deleted and then entered again. For more information on the BCMAP command, refer to the BCAM manuals.
8.4 Diagnostics

The following documents are required for diagnostic purposes:

- an exact description of the error situation
- a specification of which software was implemented with which versions
- exact specification of the system type
- the CPI-C program as the source
- the side information file (upicfile)
- the UPIC log file and the UPIC trace files; see following sections
- the CMX or PCMX trace files
- with Unix systems the core files with accompanying phases

Additional UTM documents are required for errors relating to the UTM partner application:

- KDCDEF generation and UTM diagnostics dump of the UTM partner application
- any output logs are sent to the standard output or standard error output

- Windows and Unix systems: stderr, stdout
- BS2000/OSD: SYSLST, SYSLOG, SYSOUT.

8.4.1 UPIC log file

To simplify diagnosis, the UPIC carrier system keeps a log file. A UTM error message is written to this file if the UTM application terminates a conversation abnormally. The log file is opened only for writing the error message (append mode) and is then closed again.

The file can be read using any editor.

Windows systems

The log file has the name \texttt{UPICLTid.UPL}, where \texttt{tid} is the thread ID. You can define which directory the log file will be stored in by means of the environment variable UPICLOG under the Registry key UPICW32DLL or in the \texttt{UPIC.INI} file (see section “Runtime environment, environment variables” on page 350f).

If the UPICLOG environment variable is not set, the following are interpreted in the order shown:

- the registry key UPICW32DLL
- the UPIC.INI file (not for INI-File Mapping)
- the TEMP variable
- the TMP variable
If a corresponding entry is found, the directory specified there is taken. If nothing is found, the file is stored in the \USR\TMP directory. This directory must exist and the CPI-C program must have write permission for this directory because otherwise log files will be lost.

Unix systems

The name of the log file is UPICLpid, where pid is the process ID. You use the UPICLOG shell variable to define the directory in which the log file is stored. If this shell variable is not set, the file is stored in the /usr/tmp directory.

BS2000/OSD

The name of the logging file is UPICLtsn, where tsn is the TSN of the BS2000 task.

You specify the prefix for the logging file using the job variable with the link name UPICLOG (see section “Runtime environment, linking, starting” on page 347).

If UPICLOG is not set, the system writes to the following logging file:

##.usr.tmp.UPICLtsn

If a UPIC process is re-started in BS2000 without performing a LOGOFF/LOGON, the TSN tsn is retained. This means that the logging file is overwritten!

8.4.2 UPIC trace

With the UPIC carrier system it is possible to create trace information for all CPI-C interface calls. This is controlled by setting the variable UPICTRACE.

The contents of the variable are evaluated when Enable_UTM_UPIC is called. If the variable is set, the parameters and user data up to a length of 128 bytes are logged to a file for a specific process each time a function is called. Logging is deactivated with the Disable_UTM_UPIC call.

If a CPI-C call returns a code other than CM_OK or CM_DEALLOCATED_ABEND, the cause of the error is also logged to the UPIC trace file. This provides detailed information on a specific return code for troubleshooting.
Activating the UPIC trace

You activate the UPIC trace by setting the UPICTRACE variable accordingly. The UPIC trace is activated on the individual platforms as follows:

- **Windows systems:**
  - There are different ways of activating the UPIC trace:
    - By making the appropriate setting for the UPICTRACE environment variable.
    - By ensuring that the UPICTRACE value in the Registry key UPICW32DLL contains the appropriate data.
    - By making the appropriate entry in the UPIC.INI file (see page 350). The new value takes effect immediately, i.e. logging begins the next time a CPI-C program is started.

  If the environment variable UPICTRACE is set, the value of the environment variable is used.

  If the environment variable is not set, the system checks the Registry to see whether the value UPICTRACE exists under the UPICW32DLL key and whether the corresponding data field has been filled out.

  If the environment variable is not set and the Registry entry UPICTRACE is not present, the entry in UPIC.INI is evaluated if it is present.

  The following options are available for UPICTRACE:

  \[
  \text{UPICTRACE}=-\text{S}[X] \ [-r \ \text{wrap}] \ [-d \ \text{pathname}] \\
  \]

- **Unix systems:**
  - The UPIC trace is activated when the UPICTRACE environment variable is set as follows:

  \[
  \text{UPICTRACE}=-\text{S}[X] \ [-r \ \text{wrap}] \ [-d \ \text{pathname}] \\
  \text{export UPICTRACE} \\
  \]

- **BS2000/OSD:**
  - The UPIC trace is activated as follows:

  \[
  /\text{SET-JV-LINK \ LINK-NAME=\*UPICTRA,JV-NAME=UPICTRACE} \\
  /\text{MODIFY-JV \ JV[-CONTENTS]=UPICTRACE,SET-VALUE=\-'S}[X] \ [-r \ \text{wrap}] \\
  [\text{-D} \ \text{pathname}] \\
  \]

  The -D option must be entered as an uppercase letter.
The options have the following meaning:

- **S** Full logging of the CPI-C calls, their arguments, and user data with a maximum length of 128 bytes (mandatory specification).

- **SX** An additional trace of internal information at the interface to the transport system is also provided (see also “Extended UPIC trace” on page 367). It is advisable always to use this option since problems that arise are frequently related to the transport interface.

- **r wrap**
  
  The decimal number *wrap* specifies the maximum size of the temporary trace file.

  Maximum value of *wrap*: 128
  Default value of *wrap*: 128

- **pathname / -D**
  
  The path name can be specified with blanks. If blanks are used, then the path name must be enclosed in double quotes. Double quotes can also be used if there are no blanks in the path name.

  **Windows systems:**
  The trace files are set up in the directory specified with *pathname*.
  If you do not specify **-d pathname**, the trace files are set up in the directory entered in the `TEMP` variable. If no value has been set for `TEMP`, the system attempts to do the same with `TMP`. If neither of the variables is set, the trace files will be stored in the `\USR\TMP` directory. This directory must exist and the CPI-C program must have write access to it, otherwise the trace files are lost.

  **Unix systems:**
  The trace files are set up in the directory specified with *pathname*.
  If you do not specify **-d pathname**, the trace files are set up in the `/usr/tmp` directory. The CPI-C program must have write access to this directory, otherwise the trace files are lost.

  **BS2000/OSD:**
  A file name prefix is specified for the trace files. This prefix should contain no spaces.
  If you do not specify **-D**, the names of the trace files are prefixed with `##.usr.tmp...`.
  The trace files are stored under the ID under which the program was started. The CPI-C program must be able to open the file, otherwise the trace data will be lost.

  **Example**
  If **-DTRC** is specified, the trace file `TRC.UPICTsn` will be written.
Trace files

The trace information is stored in a temporary file. This file is set up when
`Enable_UTM_UPIC` is called, and remains open until `Disable_UTM_UPIC` is called. The maximum size of this temporary file is defined by the decimal number `wrap`.

Data is logged in the file until the value (`wrap * BUFSIZ`) bytes (`BUFSIZ` as in `stdio.h`) is exceeded. A second temporary file is then created and handled in the same way.

Each time the value (`wrap * BUFSIZ`) bytes is exceeded in the current file, the trace switches to the other file. The old contents of this file are thus overwritten.

The file names of the trace files are platform-specific. The following file names have been allocated:

<table>
<thead>
<tr>
<th>Name of the file</th>
<th>Windows systems</th>
<th>Unix systems if threads are used in programs</th>
<th>BS2000/OSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st file</td>
<td>UPICTid.upt</td>
<td>UPICTpid2</td>
<td>UPICTid</td>
</tr>
<tr>
<td>2nd file</td>
<td>UPICTrid.upt</td>
<td>UPICTpid</td>
<td>UPICTid</td>
</tr>
</tbody>
</table>

1. tid = Thread ID
2. pid = Process ID
3. tsn = TSN Number

Extended UPIC trace

In an extended UPIC trace, internal information is logged at the interface to the transport system (UPIC <-> CMX or PCMX) in addition. As well as the UPIC calls, the associated CMX calls are also logged. The extended trace is structured as follows:

After logging of a UPIC call, first of all a line containing the additional plain text is output. This is followed by the logging in two lines of the last CMX functions to be called. The information is separated by a comma or `<newline>`.

1st line:

The first line contains the following information:

- Name of the CMX function called.
- Return code of the CMX function `t_error`. The return code is a hexadecimal number. If it is not zero, you can take the cause of any error which occurred from the return code.

The hexadecimal number can be decoded as follows:

- with the command `cmxdec -d 0x`hexadecimalnumber`
- using the Windows program **Trace Control** in the PCMX program window. Choose the **Error Decoding** command from the **Options** menu.
● Return code of the CMX function as decimal number (if the CMX function returns an int value).

An important exception is the CMX function `t_event`. Its return value (i.e. the event that occurred) is always output in the first column of the second line.

2nd line:

The second line logs a CMX call which was issued because of an event (`t_event`) that occurred in connection with the CMX function logged in the 1st line. The 2nd line contains the following information in the order given:

- Name of the event returned by the `t_event` function.
- Name of the CMX function called.
- Return code of `t_error` if an error occurred during the second CMX function. If applicable, it returns the reason for a connection shutdown. The number can be decoded with `cmxdec` as described above. The value “-1” denotes that there is no reason for a connection shutdown.
- The last comma in this line can be followed by a UPIC return code.

If no other CMX function was called in connection with the CMX function logged in the 1st line, only a blank and a zero are output in the 2nd line.
Deactivating the UPIC trace

You can deactivate the UPIC trace by not setting a parameter for the UPICTRACE variable.

- **Windows systems:**
  - by issuing the following SET command:
    ```
    SET UPICTRACE=
    ```
  - by modifying the entry for UPICTRACE in the `UPIC.INI` file as follows:
    ```
    UPICTRACE=
    ```
    The trace is then deactivated as soon as the CPI-C program is terminated.
  - by deleting the value UPICTRACE under the UPICW32DLL key or by simply clearing the data field for the UPICTRACE value.

- **Unix systems:**
  ```
  UPICTRACE=
  export UPICTRACE
  ```

- **BS2000/OSD:**
  - with the command
    ```
    /MODIFY-JV JV[-CONTENTS]=UPICTRACE,SET-VALUE=''
    ```
    The JV contents are deleted.
  - with the command `/DELETE-JV`
    The complete JV is deleted.

  The trace is disabled when a UPIC process is restarted.

Editing the UPIC trace

The trace information is already in printable form and does not need to be edited by a utility. Each action is logged with a time stamp and the values transferred.
8.4.3 PCMX diagnostics (Windows systems)

PCMX diagnostics are controlled by the program cmxtrace.exe. You can call this program
in the Windows program group PCMX-32 by double-clicking on the Trace Control symbol.
This program enables you to:

– activate and deactivate PCMX traces
– view PCMX traces on screen or print them out
– decode PCMX error codes (“Error Decoding” option)

The online help for the PCMX program group provides a more detailed description of how
the program works.
9 Examples

This chapter contains notes on the sample programs supplied and some simple generation examples for linking a CPI-C application on Windows systems with openUTM under BS2000/OSD, Unix systems and Windows systems.

9.1 Sample programs for Windows systems

The openUTM client for the UPIC carrier system is supplied with the following sample programs:

- **uptac** Complete CPI-C application program
- **utp32** Program for the interactive entry of individual CPI-C calls.
- **UPMULTI** CPI-C program for multiple conversations (for Windows systems only).
- **UPLTAC32** CPI-C program which loads UPIC libraries dynamically.
- **tpcall** Complete XATMI program
- **OCXTestContainer** Visual Basic client application that uses the ActiveX control element *UpicB.ocx*.
- **UpicAutClt** Visual Basic client application that uses the ActiveX automation server *UpicAutSvr.exe*.
- **UpicAutSvr** Visual Basic project with the sources of the Upic ActiveX automation server. The corresponding executable file *UpicAutSvr.exe* is stored in the installation directory under *sys*.
- **upic-cob** A Cobol project
- **UpicSimpleClient** Complete CPI-C program in Visual Basic V6.0.

In addition, the local definition file *tpcall.ldf* is provided, from which the tool XTGEN32 or XATMIGEN creates a local configuration file for the XATMI program *tpcall*. 
Sample programs (Windows)  Examples

Uptac, utp32, tpcall, UpicAutClt are ready to run after a minimum of preparation. To call them, double-click, for example, on the corresponding icons which appear in the openUTM-Client program window after installation. No separate program icon is created for OCXTestContainer because this program is largely equivalent to the UpicAutClt program.

All sample client programs are designed to be able to communicate with the sample UTM application on the server side. For more information, please refer to the README file for the UTM sample application.

The following sections provide a brief introduction to these sample programs and describe the preparations you must make to execute them.
9.1.1 uptac

`uptac` is a simple CPI-C application program. It consists of the files listed in the table below, which are stored in the directory `upic-dir\samples` after installation:

<table>
<thead>
<tr>
<th>File name</th>
<th>Type of file</th>
</tr>
</thead>
<tbody>
<tr>
<td>uptac32.c</td>
<td>C source code for the program; can be printed out</td>
</tr>
<tr>
<td>uptac32.dsp</td>
<td>Microsoft Visual C++ project file for creating an “.exe” file</td>
</tr>
<tr>
<td>uptac32.exe</td>
<td>Executable <code>uptac</code> program</td>
</tr>
<tr>
<td>uptac32.bat</td>
<td>Batch file for <code>uptac32.exe</code></td>
</tr>
</tbody>
</table>

You must configure UPIC to enable `uptac` to communicate with the UTM sample application, e.g. the following entries can be made in the `upicfile` and likewise in the TNS database (see the model entries in the `upic-dir\UPICFILE.SMP` and `upic-dir\Tnsentry.smp` files, which are also supplied):

- **Side information file:**
  - LN.DEFAULT UPIC0000
  - SD.DEFAULT SMP30111 `unixhost` PORT=30111

- **TNS entry (can be created but is no longer needed):**
  - UPIC0000\`
    - TSEL RFC1006 T’UPIC0000’ ; local name TNS
  - SMP30111.`unixhost`
    - TA RFC1006 `unixhost` PORT 30111 T’SMP30111’ ; partner_LU_name TNS

`unixhost` is the symbolic name of the host on which the UTM sample application is to run. If you want UPTAC to communicate with another UTM application, (e.g. in BS2000), you must adapt all the entries accordingly, with the exception of `LN.DEFAULT`.

In the transport address (TA...), you can also enter the Internet address of the Unix system host in place of the symbolic name. If you do so, check to ensure that the port number 30111 and the T-selector SMP30111 are also entered on the server side.

9.1.2 utp32

`utp32` is an example of a Visual Basic client application, which allows you to handle communication step by step via the CPI-C interface. To do this you enter individual CPI-C calls and the associated parameters interactively in a dialog box. The corresponding code is returned for each call.
9.1.3 **tpcall**

*tpcall* is a simple XATMI application program which allows you to implement a synchronous request/response with the sample UTM application. *tpcall* consists of the files listed in the following table, which are stored in the directory `xatmi-dir\Samples` after installation.

<table>
<thead>
<tr>
<th>File name</th>
<th>Type of file</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpcall.c</td>
<td>C source code for the program; can be printed out</td>
</tr>
<tr>
<td>tpcall32.dsp</td>
<td>Microsoft Visual C++ project file for creating an “.exe” file</td>
</tr>
<tr>
<td>tpcall32.exe</td>
<td>Executable tpcall program</td>
</tr>
</tbody>
</table>

Before using *tpcall* to communicate with the sample application, you must first:

- make entries in the `upicfile` and in the TNS, as with *uptac* (see section “uptac” on page 373)
- create a local configuration file by clicking on the XATMGEN or XTGEN32 symbol in the openUTM-Client program window.

The supplied local definition file `xatmi-dir\Samples\tpcall.1df` is then used to create the file `xatmilcf` (in the same directory).

If you want *tpcall* to be able to communicate with other applications, you may have to make changes to the `upicfile` and, hence, to the local definition file *tpcall.1df* (SVCU ... DEST statement, see also section “Configuring UPIC” on page 303).

9.1.4 **upmulti**

*upmulti* is a CPI-C application program that uses multithreading. UPMULTI establishes up to 3 conversations in parallel by creating two further threads in addition to the main thread of the process. The conversations can be established with different UTM servers or the same UTM server.

*upmulti* consists of the `upmulti.c` file, which can be found in the `upic-dir\samples` directory after installation. `upmulti.c` is the C source code for the CPI-C application program. UPMULTI. `upmulti.c` can be printed out and edited.

To enable *upmulti* to communicate with a remote UTM application, you must compile and link the program and configure it as described in chapter “Configuration” on page 317.
9.1.5 OCXTestContainer

The OCXTestContainer directory contains a Visual Basic client application. You can try out the properties and methods of the ActiveX control element UpicB.ocx using text boxes and buttons (see chapter 3 on page 35ff). The directory is located in the installation directory under samples\UpicCOM.

To allow TestContainer.exe to communicate with a remote UTM application, you must configure it as described in chapter “Configuration” on page 317.

9.1.6 UpicAutClt

The UpicAutClt directory contains a Visual Basic client application with the same interface and functionality as OCXTestContainer. You can try out the properties and methods of the ActiveX control element UpicB.ocx using text boxes and buttons. Unlike OCXTestContainer, however, UpicAutClt does not use UpicB.ocx directly but via the external ActiveX automation server UpicAutSvr.exe, which is automatically started when UpicAutClt.exe is started.

The directory is located in the installation directory under samples\UpicCOM.

To allow UpicAutClt.exe to communicate with a remote UTM application, you must configure it as described in chapter “Configuration” on page 317.

9.1.7 UpicAutSvr

The UpicAutSvr directory contains a Visual Basic project with the source codes of the external Upic ActiveX automation server. It is located in the installation directory under samples\UpicCOM. The corresponding .exe file is stored in the installation directory under sys.

With this automation module, the services of UpicB.ocx can be used not only by client programs such as UpicAutClt but also within standard Windows applications like Microsoft Office. So for Excel diagrams or standard Word letters, for example, you can implement automatic access to data which is provided by UTM server applications (see also section “The ActiveX automation server UpicAutSvr.exe” on page 68).

9.1.8 upic-cob

The directory contains a sample project to create a UPIC-Cobol application. The example was developed using a MicroFocus Cobol compiler.
9.1.9 UpicSimpleClient

The UpicSimpleClient directory contains a Visual Basic V6.0 project. There you will find the executable file UpicSimpleClient.exe and all files needed for a Visual Basic project so that you can create the UpicSimpleClient.exe file yourself.
9.2 Generation UPIC on Windows systems <-> openUTM on BS2000/OSD

The following generation example explains the principle of generating a link between a CPI-C application in Windows systems and openUTM on BS2000/OSD. Linking via RFC1006 is shown here.

In the example, the Windows system has the symbolic host name HOST123; the BS2000 host has the name HOST456.

The TNS generation is only shown for comparison purposes, since it is no longer required after V5.0.

9.2.1 Generation on the Windows system

UPIC parameters:

Enable_UTM_UPIC "UPICTTY"
Initialize_Conversation "sampladm"

Side information file C:\UPIC\UPICFILE:

* UTM(BS2000) application
SDsampladm UTMUPICR.HOST456 KDCHELP
* or. if automatic conversion of the user data
* is required
HDSampladm UTMUPICR.HOST456 KDCHELP

TNS entries in tnsxfrm format:

UPICTTY\TSEL RFC1006 T'UPICTTY'; local name RFC1006
UTMUPICR.HOST456\TA RFC1006 HOST456 PORT 102 T'UTMUPICR'; partner name RFC1006
9.2.2 Generation on the BS2000 host

In the example, HOST123 is the name of the PC as the remote system, which must be entered statically in the BCAM-RDF (resource definition file) or dynamically via the BCIN.

KDCDEF generation for the UTM(BS2000) application

| W | BCAMAPPL UTMUPICR, T-PROT=ISO |
| W | PTERM UPICTTY, PTYPE=UPIC-R, LTERM=UPIC, |
| W | LTERM UPIC, USER=UPICUSER |
| W | USER UPICUSER, STATUS=ADMIN |
9.3 Generation UPIC on Windows systems <-> openUTM on Unix systems

The following generation example explains the principle of generating a link between a CPI-C application in Windows systems and openUTM on Unix systems. Linking via RFC1006 is shown here.

In the example, the Windows system has the symbolic host name HOST123; the Unix system host has the name HOST789.

The TNS generation is only shown for comparative purposes, as it is no longer required as of Version 5.0.

9.3.1 Generation on the Windows system

UPIC parameters

Enable_UTM_UPIC "UPIC0000"  
Initialize_Conversation "sampladm"

Side information file C:\UPIC\UPICFILE

* UPIC application on Windows system  
  LNUPIC0000 UPICTTY

* partner RFC1006  
  SDsampladm UTMUPICR.HOST789 KDCHELP PORT=1230

TNS entries in tnsxfrm format

RFC1006 address format:

UPICTTY\  
  TSEL RFC1006 T'UPICTTY' ; local name RFC1006  
  TSEL LANINET A'4711' ; local name

UTMUPICR.HOST789\  
  TA RFC1006 HOST789 PORT 1230 T'UTMUPICR' ; partner RFC1006

HOSTS file

HOST789 is mapped to the Internet address in the file win-dir\HOSTS:

internetaddress HOST789

where win-dir stands for the Windows installation directory, e.g.

C:\winnt\system32\drivers\etc.
9.3.2 Generation on the Unix system

**KDCDEF generation for the UTM application on the Unix system**

- `BCAMAPPL UTMPICR`
- `PTERM UPICTTY, PTYPE=UPIC-R, LTERM=UPIC, BCAMAPPL=UTMPICR, PRONAM=HOST123`
- `LTERM UPIC, USER=UPICUSER`
- `USER UPICUSER, STATUS=ADMIN`
10 Appendix

This chapter contains the following information:

– differences from the X/Open CPI-C interface
– character set tables
– state tables

10.1 Differences vis à vis the X/Open CPI-C interface

This section describes all the extensions and special features of CPI-C with the UPIC carrier system compared to the X/Open CPI-C interface.

Extensions compared to CPI-C

● The following additional UPIC-specific functions are offered. These are:

   Enable_UTM_UPIC
   Extract_Client_Context
   Extract_Communication_Protocol
   Extract_Conversation_Encryption_Level
   Extract_Cursor_Offset
   Extract_Conversion
   Extract_Secondary_Return_Code
   Extract_Transaction_State
   Disable_UTM_UPIC
   Set_Allocate_Timer
   Set_Client_Context
   Set_Communication_Protocol
   Set_Conversation_Encryption_Level
   Set_Conversation_New_Password
   Set_Conversion
   Set_Function_Key
   Set_Partner_Host_Name
   Set_Partner_IP_Adress
   Set_Partner_Port
   Set_Partner_Tsel
The `Set_Partner_Tsel_Format` function regulates the signing on and signing off of CPI-C programs with the UPIC carrier system. If these two calls are not used, it is not possible to link to a UTM application. For further details, see section “CPI-C calls in UPIC” on page 132 and chapter “Configuration” on page 317.

- With UPIC, the `Send_Mapped_Data` and `Receive_Mapped_Data` calls are used to send and receive format names.

- Automatic conversion of user data by configuration

  This also allows for the possibility of automatic code conversion of user data between ASCII and EBCDIC code; see also section “Code conversion” on page 104. On the one hand, this reduces the effort involved in creating an application, while on the other hand it enables a single CPI-C program to communicate both with a UTM application on a Unix system based on ASCII code and with a UTM application on BS2000/OSD based on EBCDIC code (if the user data does not contain any binary information that would be corrupted in the code conversion process).

### Special features of CPI-C implementation

- The name for `partner_LU_name` can be up to 32 characters long; for a local connection via UPIC local (Unix system, Windows system) it can only be up to 8 characters.

- The name for `TP_name` can be up to 8 characters long.

### Migration from X/Open CPI-C Version 1 to X/Open CPI-C Version 2

X/Open CPI-C Specification Version 2 contains some changes compared to the previous CPI-C version. These changes also affect CPI-C programs with the UPIC carrier system, as they have been transferred.

The following changes relate to CPI-C applications in C:

- CPI-C Version 2 defines all functions of type void. Programs that query the return code must query the parameter `CM_RETURN_CODE`.

- In X/Open CPI-C Version 2, some parameters have different types than they did before. Some compilers could output warnings if existing CPI-C programs were compiled with the new CPI-C Version 2 header file.
An overview is provided in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Original X/Open CPI-C</th>
<th>X/Open CPI-C Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversation ID parameter</td>
<td>Conversation_ID (char [8])</td>
<td>unsigned char CM_PTR (unsigned char *)</td>
</tr>
<tr>
<td>Character pointers</td>
<td>char *</td>
<td>unsigned char CM_PTR (unsigned char *)</td>
</tr>
<tr>
<td>Length parameters</td>
<td>int *</td>
<td>CM_INT32 CM_PTR (signed long int *)</td>
</tr>
<tr>
<td>Definition of return codes and</td>
<td>typedef enum</td>
<td>#define</td>
</tr>
<tr>
<td>numeric parameters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 15: Modified parameters for X/Open version 2

Implementation of existing CPI-C programs with openUTM-Client as of V5.0 is object-code-compatible. To be able to use existing programs without source code modifications (determined by the migration from X/Open Version 1 to X/Open Version 2), the UPIC carrier system offers the following functionality:

- The header file contains `#defines` that have been specifically tailored to the CPI-C interface of UPIC.
- When compiling, you must set special compiler options (preprocessor symbols).

The compiler option `UTM_UPIC_V11` deactivates the X/Open-compliant part of the header file and activates the old definitions (i.e. without the security functions `Prepare_To_Receive`, `Set_Receive_Timer`, `Set_Function_Key`, `Send_Mapped_Data`, `Receive_Mapped_Data` and `Set_Receive_Type`). Without this compiler option, the opposite applies.

Unix systems:
The compiler option `UTM_UPIC_V11` must not be set in 64-bit systems.

Windows systems:
When compiling CPI-C programs on Windows systems you **must** set the compiler option `UTM_ON_WIN32`. The header file `UPIC.H` shows the effect of this option. Sie befindet sich im Verzeichnis `upic-dir\include`, `upic-dir` ist das Installationsverzeichnis von openUTM-Client mit Trägersystem UPIC

Note that compiler option `UTM_UPIC_V11` **must not** be used together with the compiler option `UTM_ON_WIN32`. If these two compiler options are set together, the program cannot be executed.

- The function prototypes are offered for ANSI compilers and K&R compilers. The usual `__STDC__` activates ANSI.
Existing CPI-C programs coded according to CPI-C Version 1 are object-code compatible with openUTM-Client V6.0. The compatibility will not be guaranteed for future versions.

### 10.2 Character sets

At the CPI-C interface, the contents of the variable `sym_dest_name` can only comprise characters from a predefined character set.

The character sets and their assignment to the variables are described below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Character set</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sym_dest_name</code></td>
<td>Set 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character</th>
<th>Character set Set 1</th>
<th>Character set Set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>;</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>::</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>“</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>a-z</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>A-Z</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0-9</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 16: Character sets
### T.61 character set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>¤</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>’</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SS2</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>PLD</td>
<td>CSI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>l</td>
<td>l</td>
<td>PLU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>SS3</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>LS1</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>LS0</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>-</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 17: Code table T.61 in accordance with CCITT recommendation

#### Meaning of abbreviations:

BS= BACKSPACE  
SUB= SUBSTITUTE CHARACTER  
LF= LINE FEED  
ESC= ESCAPE  
FF= FORM FEED  
SS3= SINGLE-SHIFT THREE  
CR= CARRIAGE RETURN  
SP= SPACE  
LS1= LOCKING SHIFT ONE  
PLD= PARTIAL LINE DOWN  
LS0= LOCKING SHIFT ZERO  
PLU= PARTIAL LINE UP  
SS2= SINGLE-SHIFT TWO  
CSI= CONTROL SEQUENCE INTRODUCER

Table 18: Abbreviations of special characters
### 10.3 State table

In the following table, the follow-up state of a program that was previously in a particular state is indicated for the individual calls (depending on their result). An explanation of the abbreviations used in the table is then provided.

<table>
<thead>
<tr>
<th>Call</th>
<th>Result</th>
<th>Follow-up state, if previously in state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Start</td>
</tr>
<tr>
<td>Initialize_Conversation</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Initialize_Conversation</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Initialize_Conversation</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Allocate</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Allocate</td>
<td>ae</td>
<td>psc</td>
</tr>
<tr>
<td>Allocate</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Allocate</td>
<td>pe</td>
<td>psc</td>
</tr>
<tr>
<td>Allocate</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Deallocate</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Deallocate</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Deferred_Deallocate</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Extract_Client_Context</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Client_Context</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Client_Context</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Communication_Protocol</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Communication_Protocol</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Communication_Protocol</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversation_Encryption_Level</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversation_Encryption_Level</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversation_Encryption_Level</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversation_State</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversation_State</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversation_State</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversion</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Conversion</td>
<td>pc</td>
<td>psc</td>
</tr>
</tbody>
</table>

Table 19: State table for CPI-C calls
## State table

<table>
<thead>
<tr>
<th>Call</th>
<th>Result</th>
<th>Follow-up state, if previously in state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Start</td>
</tr>
<tr>
<td>Extract_Conversion</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Cursor_Offset</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Cursor_Offset</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Cursor_Offset</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Partner_LU_Name</td>
<td>ok</td>
<td>-</td>
</tr>
<tr>
<td>Extract_Partner_LU_Name</td>
<td>pc</td>
<td>-</td>
</tr>
<tr>
<td>Extract_Partner_LU_Name</td>
<td>ps</td>
<td>-</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>nr</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Secondary_Return_Code</td>
<td>ps</td>
<td>-</td>
</tr>
<tr>
<td>Extract_Transaction_State</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Transaction_State</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Extract_Transaction_State</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Prepare_To_Receive</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Prepare_To_Receive</td>
<td>da</td>
<td>psc</td>
</tr>
<tr>
<td>Prepare_To_Receive</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Prepare_To_Receive</td>
<td>rf</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>ok{dr,no}</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>ok{nd,se}</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>ok{dr,se}</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>ae</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>da</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>dn</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>rf</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>oi,un</td>
<td>psc</td>
</tr>
</tbody>
</table>

Table 19: State table for CPI-C calls
<table>
<thead>
<tr>
<th>Call</th>
<th>Result</th>
<th>Follow-up state, if previously in state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Start</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Receive / Receive_Mapped_Data</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Send_Data / Send_Mapped_Data</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Send_Data / Send_Mapped_Data</td>
<td>ae</td>
<td>psc</td>
</tr>
<tr>
<td>Send_Data / Send_Mapped_Data</td>
<td>da</td>
<td>psc</td>
</tr>
<tr>
<td>Send_Data / Send_Mapped_Data</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Send_Data / Send_Mapped_Data</td>
<td>rf</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Allocate_Timer</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Allocate_Timer</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Allocate_Timer</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Client_Context</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Client_Context</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Client_Context</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Communication_Protocol</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Communication_Protocol</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Communication_Protocol</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Encryption_Level</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Encryption_Level</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Encryption_Level</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversion</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversion</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversion</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_Type</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_Type</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_Type</td>
<td>pn</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_New_Password</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_New_Password</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_Password</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_Password</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Conversation_Security_User_ID</td>
<td>ok</td>
<td>psc</td>
</tr>
</tbody>
</table>

Table 19: State table for CPI-C calls
### State table

<table>
<thead>
<tr>
<th>Call</th>
<th>Result</th>
<th>Follow-up state, if previously in state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Start</td>
<td>Reset</td>
</tr>
<tr>
<td>Set_Conversation_Security_User_ID</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Deallocate_Type</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Deallocate_Type</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Deallocate_Type</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Function_Key</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Function_Key</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Function_Key</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Receive_Time</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Receive_Time</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Receive_Time</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Receive_Type</td>
<td>ok</td>
<td>-</td>
</tr>
<tr>
<td>Set_Receive_Type</td>
<td>pc</td>
<td>-</td>
</tr>
<tr>
<td>Set_Partner_Host_Name</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Host_Name</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Host_Name</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_IP_Address</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_IP_Address</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_IP_Address</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_LU_Name</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_LU_Name</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_LU_Name</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Port</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Port</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Tsel</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Tsel</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Tsel</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Tsel_Format</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Tsel_Format</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Partner_Tsel_Format</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set.Sync_Level</td>
<td>ok</td>
<td>psc</td>
</tr>
</tbody>
</table>

Table 19: State table for CPI-C calls
<table>
<thead>
<tr>
<th>Call</th>
<th>Result</th>
<th>Follow-up state, if previously in state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Start</td>
</tr>
<tr>
<td>Set_Sync_Level</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_Sync_Level</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Set_TP_Name</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Set_TP_Name</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Set_TP_Name</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Port</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Port</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Port</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Tsel</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Tsel</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Tsel</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Tsel_Format</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Tsel_Format</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Local_Tsel_Format</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Secondary_Return_Code</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Secondary_Return_Code</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Specify_Secondary_Return_Code</td>
<td>ps</td>
<td>psc</td>
</tr>
<tr>
<td>Enable_UTM_UPIC</td>
<td>ok</td>
<td>Reset</td>
</tr>
<tr>
<td>Enable_UTM_UPIC</td>
<td>pc</td>
<td>-</td>
</tr>
<tr>
<td>Enable_UTM_UPIC</td>
<td>ps</td>
<td>-</td>
</tr>
<tr>
<td>Disable_UTM_UPIC</td>
<td>ok</td>
<td>psc</td>
</tr>
<tr>
<td>Disable_UTM_UPIC</td>
<td>pc</td>
<td>psc</td>
</tr>
<tr>
<td>Disable_UTM_UPIC</td>
<td>ps</td>
<td>psc</td>
</tr>
</tbody>
</table>

Table 19: State table for CPI-C calls

1 Permitted only directly after a \textit{Receive/Receive\_Mapped\_Data} call
### Abbreviations for the state table:

<table>
<thead>
<tr>
<th>Result</th>
<th>Return codes</th>
</tr>
</thead>
</table>
| ae     | CM_ALLOCATE_FAILURE_RETRY  
|        | CM_ALLOCATE_FAILURE_NO_RETRY  
|        | CM_SECURITY_NOT_VALID  
|        | CM_SECURITY_NOT_SUPPORTED  
|        | CM_TPN_NOT_RECOGNIZED  
|        | CM_TP_NOT_AVAILABLE_NO_RETRY  
|        | CM_TP_NOT_AVAILABLE_RETRY  
| da     | CM_DEALLOCATED_ABEND  
| dn     | CM_DEALLOCATED_NORMAL  
| oi     | CM_OPERATION_INCOMPLETE  
| ok     | CM_OK  
| pe     | CM_PARAMETER_ERROR  
| pc     | CM_PROGRAM_PARAMETER_CHECK  
| pn     | CM_PARAM_VALUE_NOT_SUPPORTED  
| ps     | CM_PRODUCT_SPECIFIC_ERROR  
| rf     | CM_RESOURCE_FAILURE_RETRY  
|        | CM_RESOURCE_FAILURE_NO_RETRY  
| nr     | CM_NO_SECONDARY_RETURN_CODE  
| un     | CM_OPERATION_UNSUCCESSFUL  

Table 20: Abbreviations for the state table (1)

<table>
<thead>
<tr>
<th>Result</th>
<th>data_received and status_received:</th>
</tr>
</thead>
</table>
| dr     | CM_COMPLETE_DATA_RECEIVED  
|        | CM_INCOMPLETE_DATA_RECEIVED  
| nd     | CM_NO_DATA_RECEIVED  
| no     | CM_NO_STATUS_RECEIVED  
| se     | CM_SEND_RECEIVED  

Table 21: Abbreviations for the state table (2)
<table>
<thead>
<tr>
<th>Follow-up state</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>No state change</td>
</tr>
<tr>
<td>psc</td>
<td>Error CM_PROGRAM_STATE_CHECK</td>
</tr>
</tbody>
</table>

Table 22: Abbreviations for the state table (3)

The return code CM_CALL_NOT_SUPPORTED is not included in the state table. It is returned if the UPIC library includes the call but the function is not supported in the specific situation. There is no change of state.
Glossary

A term in italic font means that it is explained somewhere else in the glossary.

abnormal termination of a UTM application
Termination of a UTM application, where the KDCFILE is not updated. Abnormal termination is caused by a serious error, such as a crashed computer or an error in the system software. If you then restart the application, openUTM carries out a warm start.

abstract syntax (OSI)
Abstract syntax is defined as the set of formally described data types which can be exchanged between applications via OSI TP. Abstract syntax is independent of the hardware and programming language used.

acceptor (CPI-C)
The communication partners in a conversation are referred to as the initiator and the acceptor. The acceptor accepts the conversation initiated by the initiator with Accept_Conversation.

access list
An access list defines the authorization for access to a particular service, TAC queue or USER queue. An access list is defined as a key set and contains one or more key codes, each of which represent a role in the application. Users or LTERMs can only access the service or TAC queue/USER queue when the corresponding roles have been assigned to them (i.e. when their key set and the access list contain at least one common key code).

access point (OSI)
See service access point.

ACID properties
Acronym for the fundamental properties of transactions: atomicity, consistency, isolation and durability.

administration
Administration and control of a UTM application by an administrator or an administration program.
administration command
Commands used by the administrator of a UTM application to carry out administration functions for this application. The administration commands are implemented in the form of transaction codes.

administration journal
See cluster administration journal.

administration program
Program unit containing calls to the program interface for administration. This can be either the standard administration program KDCADM that is supplied with openUTM or a program written by the user.

administrator
User who possesses administration authorization.

AES
AES (Advanced Encryption Standard) is the current symmetric encryption standard defined by the National Institute of Standards and Technology (NIST) and based on the Rijndael algorithm developed at the University of Leuven (Belgium). If the AES method is used, the UPIC client generates an AES key for each session.

Apache Axis
Apache Axis (Apache eXtensible Interaction System) is a SOAP engine for the design of Web services and client applications. There are implementations in C++ and Java.

Apache Tomcat
Apache Tomcat provides an environment for the execution of Java code on Web servers. It was developed as part of the Apache Software Foundation's Jakarta project. It consists of a servlet container written in Java which can use the JSP Jasper compiler to convert JavaServer pages into servlets and run them. It also provides a fully featured HTTP server.

application context (OSI)
The application context is the set of rules designed to govern communication between two applications. This includes, for instance, abstract syntaxes and any assigned transfer syntaxes.
application entity (OSI)
An application entity (AE) represents all the aspects of a real application which are relevant to communications. An application entity is identified by a globally unique name (“globally” is used here in its literal sense, i.e. worldwide), the application entity title (AET). Every application entity represents precisely one application process. One application process can encompass several application entities.

application entity qualifier (OSI)
Component of the application entity title. The application entity qualifier identifies a service access point within an application. The structure of an application entity qualifier can vary. openUTM supports the type “number”.

application entity title (OSI)
An application entity title is a globally unique name for an application (“globally” is used here in its literal sense, i.e. worldwide). It is made up of the application process title of the relevant application process and the application entity qualifier.

application information
This is the entire set of data used by the UTM application. The information comprises memory areas and messages of the UTM application including the data currently shown on the screen. If operation of the UTM application is coordinated with a database system, the data stored in the database also forms part of the application information.

application process (OSI)
The application process represents an application in the OSI reference model. It is uniquely identified globally by the application process title.

application process title (OSI)
According to the OSI standard, the application process title (APT) is used for the unique identification of applications on a global (i.e. worldwide) basis. The structure of an application process title can vary. openUTM supports the type Object Identifier.

application program
An application program is the core component of a UTM application. It comprises the main routine KDCROOT and any program units and processes all jobs sent to a UTM application.
application restart
Restart of a UTM-S application after it has terminated abnormally. The application information is reset to the most recent consistent state. Interrupted dialog services are rolled back to the most recent synchronization point, allowing processing to be resumed in a consistent state from this point (service restart). Interrupted asynchronous services are rolled back and restarted or restarted at the most recent synchronization point. For UTM-F applications, only configuration data which has been dynamically changed is rolled back to the most recent consistent state after a restart due to a preceding abnormal termination.

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.

asynchronous job
Job carried out by the job submitter at a later time. openUTM includes message queuing functions for processing asynchronous jobs (see UTM-controlled queue and service-controlled queue). An asynchronous job is described by the asynchronous message, the recipient and, where applicable, the required execution time.
If the recipient is a terminal or a printer, the asynchronous job is a queued output job. If the recipient is an asynchronous service of the same application or a remote application, the job is a background job.
Asynchronous jobs can be time-driven jobs or can be integrated in a job complex.
asynchronous message
Asynchronous messages are messages directed to a message queue. They are stored temporarily by the local UTM application and then further processed regardless of the job submitter. Distinctions are drawn between the following types of asynchronous messages, depending on the recipient:

- In the case of asynchronous messages to a UTM-controlled queue, all further processing is controlled by openUTM. This type includes messages that start a local or remote asynchronous service (see also background job) and messages sent for output on a terminal or printer (see also queued output job).
- In the case of asynchronous messages to a service-controlled queue, further processing is controlled by a service of the application. This type includes messages to a TAC queue, messages to a USER queue and messages to a temporary queue. The USER queue and the temporary queue must belong to the local application, whereas the TAC queue can be in both the local application and the remote application.

asynchronous program
Program unit started by a background job.

asynchronous service (KDCS)
Service which processes a background job. Processing is carried out independently of the job submitter. An asynchronous service can comprise one or more program units/transactions. It is started via an asynchronous transaction code.

audit (BS2000/OSD)
During execution of a UTM application, UTM events which are of relevance in terms of security are logged by SAT for auditing purposes.

authentication
See system access control.

authorization
See data access control.

Axis
See Apache Axis.

background job
Background jobs are asynchronous jobs destined for an asynchronous service of the current application or of a remote application. Background jobs are particularly suitable for time-intensive processing or processing which is not time-critical and where the results do not directly influence the current dialog.
basic format
Format in which terminal users can make all entries required to start a service.

basic job
Asynchronous job in a job complex.

browsing asynchronous messages
A service sequentially reads the asynchronous messages in a service-controlled queue. The messages are not locked while they are being read and they remain in the queue after they have been read. This means that they can be read simultaneously by different services.

bypass mode (BS2000/OSD)
Operating mode of a printer connected locally to a terminal. In bypass mode, any asynchronous message sent to the printer is sent to the terminal and then redirected to the printer by the terminal without being displayed on screen.

cache
Used for buffering application data for all the processes of a UTM application. The cache is used to optimize access to the page pool.

CCS name (BS2000/OSD)
See coded character set name.

client
Clients of a UTM application can be:
– terminals
– UPIC client programs
– transport system applications (e.g. DCAM, PDN, CMX, socket applications or UTM applications which have been generated as transport system applications).

Clients are connected to the UTM application via LTERM partners. openUTM clients which use the OpenCPIC carrier system are treated just like OSI TP partners.

client side of a conversation
This term has been superseded by initiator.

cluster
A number of computers connected over a fast network and which in many cases can be seen as a single computer externally. The objective of clustering is generally to increase the computing capacity or availability in comparison with a single computer.
cluster administration journal
Files used to log administrative actions performed on a UTM cluster application. The administration journal files serve to pass on to the other node applications those administrative actions that are to apply globally to all node applications in a UTM cluster application.

cluster configuration file
File containing the central configuration data of a UTM cluster application.

cluster user file
File containing the user management data of a UTM cluster application.

coded character set name (BS2000/OSD)
If the product XHCS (eXtended Host Code Support) is used, each character set used is uniquely identified by a coded character set name (abbreviation: “CCS name” or “CCSN”).

cold start
Start of a UTM application after the application terminates normally (normal termination) or after a new generation (see also warm start).

communication area (KDCS)
KDCS primary storage area, secured by transaction logging and which contains service-specific data. The communication area comprises 3 parts:
- the KB header with general service data
- the KB return area for returning values to KDCS calls
- the KB program area for exchanging data between UTM program units within a single service.

communication resource manager
In distributed systems, communication resource managers (CRMs) control communication between the application programs. openUTM provides CRMs for the international OSI TP standard, for the LU6.1 industry standard and for the proprietary openUTM protocol UPIC.
configuration

Sum of all the properties of a UTM application. The configuration describes:
– application parameters and operating parameters
– the objects of an application and the properties of these objects. Objects can be program units and transaction codes, communication partners, printers, user IDs, etc.
– defined measures for controlling data and system access.

The configuration of a UTM application is defined at generation time and can be changed dynamically by the administrator (while the application is running). The configuration is stored in the KDCFILE.

Also:
The process of defining the configuration of the UTM application. A distinction is made between static and dynamic configuration.

comfirmation job

Component of a job complex where the confirmation job is assigned to the basic job. There are positive and negative confirmation jobs. If the basic job returns a positive result, the positive confirmation job is activated, otherwise, the negative confirmation job is activated.

connection bundle

see LTERM bundle.

contention loser

Every connection between two partners is managed by one of the partners. The partner that manages the connection is known as the contention winner. The other partner is the contention loser.

contention winner

A connection’s contention winner is responsible for managing the connection. Jobs can be started by the contention winner or by the contention loser. If a conflict occurs, i.e. if both partners in the communication want to start a job at the same time, then the job stemming from the contention winner uses the connection.

conversation

In CPI-C, communication between two CPI-C application programs is referred to as a conversation. The communication partners in a conversation are referred to as the initiator and the acceptor.

conversation ID

CPI-C assigns a local conversation ID to each conversation, i.e. the initiator and acceptor each have their own conversation ID. The conversation ID uniquely assigns each CPI-C call in a program to a conversation.
CPI-C

CPI-C (Common Programming Interface for Communication) is a program interface for program-to-program communication in open networks standardized by X/Open and CIW (CPI-C Implementor’s Workshop). The CPI-C implemented in openUTM complies with X/Open’s CPI-C V2.0 CAE Specification. The interface is available in COBOL and C. In openUTM, CPI-C can communicate via the OSI TP, LU6.1 and UPIC protocols and with openUTM-LU62.

Cross Coupled System / XCS

Cluster of BS2000 computers with the Highly Integrated System Complex Multiple System Control Facility (HIPLEX® MSCF).

data access control

In data access control openUTM checks whether the communication partner is authorized to access a particular object belonging to the application. The access rights are defined as part of the configuration.

dead letter queue

The dead letter queue is a TAC queue which has the fixed name KDCDLETQ. It is always available to save queued messages sent to transaction codes or TAC queues but which could not be processed. The saving of queued messages in the dead letter queue can be activated or deactivated for each message destination individually using the TAC statement's DEAD-LETTER-Q parameter.

DES

DES (Data Encryption Standard) is an international standard for encrypting data. One key is used in this method for encoding and decoding. If the DES method is used, the UPIC client generates a DES key for each session.

dialog conversation

CPI-C conversation in which both the initiator and the acceptor are permitted to send. A dialog transaction code for the acceptor must have been generated in the UTM application.

dialog job, interactive job

Job which starts a dialog service. The job can be issued by a client or, when two servers communicate with each other (server-server communication), by a different application (job-submitting service).
**dialog message**
A message which requires a response or which is itself a response to a request. The request and the response both take place within a single service. The request and reply together form a dialog step.

**dialog program**
Program unit which partially or completely processes a dialog step.

**dialog service**
Service which processes a job interactively (synchronously) in conjunction with the job submitter. A dialog service processes dialog messages received from the job submitter and generates dialog messages to be sent to the job submitter. A dialog service comprises at least one transaction. In general, a dialog service encompasses at least one dialog step. Exception: in the event of service chaining, it is possible for more than one service to comprise a dialog step.

**dialog step**
A dialog step starts when a dialog message is received by the UTM application. It ends when the UTM application responds.

**dialog terminal process (Unix systems/Windows systems)**
A dialog terminal process connects a terminal of a Unix system or a Windows system with the work processes of the UTM application. Dialog terminal processes are started either when the user enters utmdtp or via the LOGIN shell. A separate dialog terminal process is required for each terminal to be connected to a UTM application.

**Distributed Lock Manager / DLM (BS2000/OSD)**
Concurrent, cross-computer file accesses can be synchronized using the Distributed Lock Manager. DLM is a basic function of HIPLEX® MSCF.

**distributed processing**
Processing of dialog jobs by several different applications or the transfer of background jobs to another application. The higher-level protocols LU6.1 and OSI TP are used for distributed processing. openUTM-LU62 also permits distributed processing with LU6.2 partners. A distinction is made between distributed processing with global transactions (transaction logging across different applications) and distributed processing without global transactions (local transaction logging only). Distributed processing is also known as server-server communication.

**distributed transaction**
See global transaction.


distributed transaction processing
X/Open architecture model for transaction-oriented distributed processing with global transactions.

dynamic configuration
Changes to the configuration made by the administrator. UTM objects such as program units, transaction codes, clients, LU6.1 connections, printers or user IDs can be added, modified or in some cases deleted from the configuration while the application is running. To do this, it is necessary to create separate administration programs which use the functions of the program interface for administration. The WinAdmin administration program can be used to do this, or separate administration programs must be created that utilize the functions of the administration program interface.

encryption level
The encryption level specifies if and to what extent a client message and password are to be encrypted.

event-driven service
This term has been superseded by event service.

event exit
Routine in an application program which is started automatically whenever certain events occur (e.g. when a process is started, when a service is terminated). Unlike event services, an event exit must not contain any KDCS, CPI-C or XATMI calls.

event function
Collective term for event exits and event services.

event service
Service started when certain events occur, e.g. when certain UTM messages are issued. The program units for event-driven services must contain KDCS calls.

generation
Static configuration of a UTM application using the UTM tool KDCDEF and creation of an application program.

global secondary storage area
See secondary storage area.

global transaction
Transaction which encompasses more than one application.
hardcopy mode
Operating mode of a printer connected locally to a terminal. Any message which is displayed on screen will also be sent to the printer.

heterogeneous link
In the case of server-server communication: a link between a UTM application and a non-UTM application, e.g. a CICS or TUXEDO application.

Highly Integrated System Complex / HIPLEX®
Product family for implementing an operating, load sharing and availability cluster made up of a number of BS2000 servers.

HIPLEX® MSCF
(MSCF = Multiple System Control Facility)
Provides the infrastructure and basic functions for distributed applications with HIPLEX®.

homogeneous link
In the case of server-server communication: a link between two UTM applications. It is of no significance whether the applications are running on the same operating system platforms or on different platforms.

inbound conversation (CPI-C)
See incoming conversation.

incoming conversation (CPI-C)
A conversation in which the local CPI-C program is the acceptor is referred to as an incoming conversation. In the X/Open specification, the term “inbound conversation” is used synonymously with “incoming conversation”.

initial KDCFILE
In a UTM cluster application, this is the KDCFILE generated by KDCDEF and which must be copied for each node application before the node applications are started.

initiator (CPI-C)
The communication partners in a conversation are referred to as the initiator and the acceptor. The initiator sets up the conversation with the CPI-C calls Initialize_Conversation and Allocate.

insert
Field in a message text in which openUTM enters current values.
inverse KDCDEF
A function which uses the dynamically adapted configuration data in the KDCFILE to generate control statements for a KDCDEF run. An inverse KDCDEF can be started “offline” under KDCDEF or “online” via the program interface for administration.

JDK
Java Development Kit
Standard development environment from Sun Microsystems for the development of Java applications.

job
Request for a service provided by a UTM application. The request is issued by specifying a transaction code. See also: queued output job, dialog job, background job, job complex.

job complex
Job complexes are used to assign confirmation jobs to asynchronous jobs. An asynchronous job within a job complex is referred to as a basic job.

job-receiving service (KDCS)
A job-receiving service is a service started by a job-submitting service of another server application.

job-submitting service (KDCS)
A job-submitting service is a service which requests another service from a different server application (job-receiving service) in order to process a job.

KDCADM
Standard administration program supplied with openUTM. KDCADM provides administration functions which are called with transaction codes (administration commands).

KDCDEF
UTM tool for the generation of UTM applications. KDCDEF uses the configuration information in the KDCDEF control statements to create the UTM objects KDCFILE and the ROOT table sources for the main routine KDCROOT.

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.
Glossary

**KDCROOT**
Main routine of an *application program* which forms the link between the *program units* and the UTM system code. KDCROOT is linked with the *program units* to form the *application program*.

**KDCS message area**
For KDCS calls: buffer area in which messages or data for openUTM or for the *program unit* are made available.

**KDCS parameter area**
See *parameter area*.

**KDCS program interface**
Universal UTM program interface compliant with the national DIN 66 265 standard and which includes some extensions. KDCS (compatible data communications interface) allows dialog services to be created, for instance, and permits the use of *message queuing* functions. In addition, KDCS provides calls for *distributed processing*.

**Kerberos**
Kerberos is a standardized network authentication protocol (RFC1510) based on encryption procedures in which no passwords are sent to the network in clear text.

**Kerberos principal**
Owner of a key. Kerberos uses symmetrical encryption, i.e. all the keys are present at two locations, namely with the key owner (principal) and the KDC (Key Distribution Center).

**key code**
Code that represents specific access authorization or a specific role. Several key codes are grouped into a *key set*.

**key set**
Group of one or more *key codes* under a particular a name. A key set defines authorization within the framework of the authorization concept used (lock/key code concept or access list concept). A key set can be assigned to a *user ID*, an *LTERM partner*, a *service* or a *TAC queue*.

**linkage program**
See *KDCROOT*. 
local secondary storage area
See secondary storage area.

Log4j
Log4j is part of the Apache Jakarta project. Log4j provides information for logging information (runtime information, trace records, etc.) and configuring the log output. WS4UTM uses the software product Log4j for trace and logging functionality.

lock code
Code protecting an LTERM partner or transaction code against unauthorized access. Access is only possible if the key set of the accesser contains the appropriate key code (lock/key code concept).

LPAP bundle
LPAP bundles allow messages to be distributed to LPAP partners across several partner applications. If a UTM application has to exchange a very large number of messages with a partner application then load distribution may be improved by starting multiple instances of the partner application and distributing the messages across the individual instances. In an LPAP bundle, openUTM is responsible for distributing the messages to the partner application instances. An LPAP bundle consists of a master LPAP and multiple slave LPAPs. The slave LPAPs are assigned to the master LPAP on generation. LPAP bundles exist for both the OSI TP protocol and the LU6.1 protocol.

LPAP partner
In the case of distributed processing via the LU6.1 protocol, an LPAP partner for each partner application must be configured in the local application. The LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned LPAP partner and not by the application name or address.

LTERM bundle
An LTERM bundle (connection bundle) consists of a master LTERM and multiple slave LTERMs. An LTERM bundle (connection bundle) allows you to distribute queued messages to a logical partner application evenly across multiple parallel connections.

LTERM group
An LTERM group consists of one or more alias LTERMs, the group LTERMs and a primary LTERM. In an LTERM group, you assign multiple LTERMs to a connection.
**LTERM partner**
LTERM partners must be configured in the application if you want to connect clients or printers to a *UTM application*. A client or printer can only be connected if an LTERM partner with the appropriate properties is assigned to it. This assignment is generally made during configuration, but can also be made dynamically using terminal pools.

**LTERM pool**
The TPOOL statement allows you to define a pool of LTERM partners instead of issuing one LTERM and one PTERM statement for each *client*. If a client establishes a connection via an LTERM pool, an LTERM partner is assigned to it dynamically from the pool.

**LU6.1**
Device-independent data exchange protocol (industrial standard) for transaction-oriented *server-server communication*.

**LU6.1-LPAP bundle**
*LPAP bundle* for *LU6.1* partner applications.

**main process (Unix systems / Windows systems)**
Process which starts the *UTM application*. It starts the *work processes*, *printer processes*, *network processes* and the *timer process* and monitors the *UTM application*.

**main routine KDCROOT**
See *KDCROOT*.

**mapped host name**
Mapping of the partner application's UTM host name to a real host name or vice versa.

**message definition file**
The message definition file is supplied with openUTM and, by default, contains the UTM message texts in German and English together with the definitions of the message properties. Users can take this file as a basis for their own message modules.

**message destination**
Output medium for a *message*. Possible message destinations for a message include, for instance, terminals, *TS applications*, the *event service MSGTAC*, the *system log file SYSLOG* or *TAC queues*, *asynchronous TACs* and *USER queues*. 
message queue
Queue in which specific messages are kept with transaction management until further processed. A distinction is drawn between service-controlled queues and UTM-controlled queues, depending on who monitors further processing.

message queuing
Message queuing (MQ) is a form of communication in which the messages are exchanged via intermediate queues rather than directly. The sender and recipient can be separated in space or time, and transfer of the message is still guaranteed, irrespective of whether a network connection is available at the time or not. In openUTM there are UTM-controlled queues and service-controlled queues.

message router (BS2000/OSD)
Device in a central host or a communication computer which distributes queued input messages to different UTM applications which can be located on different computers. The message router also allows you to work with multiplex connections.

MSGTAC
Special event service that processes messages with the message destination MSGTAC by means of a program. MSGTAC is an asynchronous service and is created by the operator of the application.

multiplex connection (BS2000/OSD)
Special method of connecting terminals to a UTM application. A multiplex connection enables several terminals to share a single transport connection.

multi-step service (KDCS)
Service carried out in a number of dialog steps.

multi-step transaction
Transaction which comprises more than one processing step.

Network File System/Service / NFS
Allows Unix systems to access file systems across the network.

network process (Unix systems / Windows systems)
A process in a UTM application for connection to the network.

network selector
The network selector identifies a service access point to the network layer of the OSI reference model in the local system.
node
Individual computer of a cluster.

node application
UTM application that is executed on an individual node as part of a UTM cluster application.

normal termination of a UTM application
Controlled termination of a UTM application. Among other things, this means that the administration data in the KDCFILE are updated. The administrator initiates normal termination (e.g. with KDCSHUT N). After a normal termination, openUTM carries out any subsequent start as a cold start.

object identifier
An object identifier is an identifier for objects in an OSI environment which is globally unique (i.e. throughout the world). An object identifier comprises a sequence of integers which represent a path in a tree structure.

open terminal pool
Terminal pool which is not restricted to clients of a single computer or particular type. Any client for which no computer- or type-specific terminal pool has been generated can connect to this terminal pool.

online import
In a UTM cluster application, online import refers to the import of application data from a normally terminated node application into a running node application.

online update
In a UTM cluster application, online update refers to a change to the application configuration or the application program or the use of a new UTM revision level while a UTM cluster application is running.

openSM2
The openSM2 product line offers a consistent solution for the enterprise-wide performance management of server and storage systems. openSM2 offers the acquisition of monitoring data, online monitoring and offline evaluation.

openUTM application
See UTM application.
openUTM cluster
From the perspective of UPIC clients, not from the perspective of the server: Combination of several node applications of a UTM cluster application to form one logical application that is addressed via a common symbolic destination name.

openUTM-D
openUTM-D (openUTM distributed) is a component of openUTM which allows distributed processing. openUTM-D is an integral component of openUTM.

OSI-LPAP bundle
LPAP bundle for OSI TP partner applications.

OSI-LPAP partner
OSI-LPAP partners are the addresses of the OSI TP partners generated in openUTM. In the case of distributed processing via the OSI TP protocol, an OSI-LPAP partner for each partner application must be configured in the local application. The OSI-LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned OSI-LPAP partner and not by the application name or address.

OSI reference model
The OSI reference model provides a framework for standardizing communications in open systems. ISO, the International Organization for Standardization, described this model in the ISO IS7498 standard. The OSI reference model divides the necessary functions for system communication into seven logical layers. These layers have clearly defined interfaces to the neighboring layers.

OSI TP
Communication protocol for distributed transaction processing defined by ISO. OSI TP stands for Open System Interconnection Transaction Processing.

OSI TP partner
Partner of the UTM application that communicates with the UTM application via the OSI TP protocol.
Examples of such partners are:
- a UTM application that communicates via OSI TP
- an application in the IBM environment (e.g. CICS) that is connected via openUTM-LU62
- an application of the OpenCPIC carrier system of the openUTM client
- applications from other TP monitors that support OSI TP
outbound conversation (CPI-C)
See outgoing conversation.

outgoing conversation (CPI-C)
A conversation in which the local CPI-C program is the initiator is referred to as an outgoing conversation. In the X/Open specification, the term “outbound conversation” is used synonymously with “outgoing conversation”.

page pool
Part of the KDCFILE in which user data such as dialog messages, asynchronous messages and secondary storage areas is stored.

parameter area
Data structure in which a program unit passes the operands required for a UTM call to openUTM.

postselection (BS2000/OSD)
Selection of logged UTM events from the SAT logging file which are to be evaluated. Selection is carried out using the SATUT tool.

predialog (BS2000/OSD)
Request from a terminal user to the data communication system to establish a virtual connection to the application. The predialog is unnecessary if the application requests the establishment of a virtual connection.

preselection (BS2000/OSD)
Definition of the UTM events which are to be logged for the SAT audit. Preselection is carried out with the UTM-SAT administration functions. A distinction is made between event-specific, user-specific and job-specific (TAC-specific) preselection.

presentation selector
The presentation selector identifies a service access point to the presentation layer of the OSI reference model in the local system.

primary storage area
Area in main memory to which the KDCS program unit has direct access, e.g. standard primary working area, communication area.

print administration
Functions for print control and the administration of queued output jobs, sent to a printer.
print control
openUTM functions for controlling print output.

printer control LTERM
A printer control LTERM allows a client or terminal user to connect to a UTM application. The printers assigned to the printer control LTERM can then be administered from the client program or the terminal. No administration rights are required for these functions.

printer control terminal
This term has been superseded by printer control LTERM.

printer group (Unix systems)
For each printer, a Unix system sets up one printer group by default that contains this one printer only. It is also possible to assign several printers to one printer group or to assign one printer to several different printer groups.

printer pool
Several printers assigned to the same LTERM partner.

printer process (Unix systems)
Process set up by the main process for outputting asynchronous messages to a printer group. The process exists as long as the printer group is connected to the UTM application. One printer process exists for each connected printer group.

process
The openUTM manuals use the term “process” as a collective term for processes (Unix systems / Windows systems) and tasks (BS2000/OSD).

processing step
A processing step starts with the receipt of a dialog message sent to the UTM application by a client or another server application. The processing step ends either when a response is sent, thus also terminating the dialog step, or when a dialog message is sent to a third party.

program interface for administration
UTM program interface which helps users to create their own administration programs. Among other things, the program interface for administration provides functions for dynamic configuration, for modifying properties and application parameters and for querying information on the configuration and the current workload of the application.
program unit
UTM services are implemented in the form of one or more program units. The program units are components of the application program. They contain UTM function calls and can be addressed using transaction codes. Several different transaction codes can be assigned to a single program unit.

queue
See message queue.

queued output job
Queued output jobs are asynchronous jobs which output a message, such as a document, to a printer or a terminal. The output destination can, however, also be another application connected via the transport system interface. Queued output jobs are processed by UTM system functions exclusively, i.e. it is not necessary to create program units to process them.

Quick Start Kit
A sample application supplied with openUTM (Windows systems).

redelivery
Repeated delivery of an asynchronous message that could not be processed correctly because, for example, the transaction was rolled back or the asynchronous service was terminated abnormally. The message is returned to the message queue and can then be read and/or processed again.

reentrant program
Program whose code is not altered when it runs. In BS2000/OSD this constitutes a prerequisite for using shared code.

request
Request from a client or another server for a service function.

requestor
In XATMI, the term requestor refers to an application which calls a service.

resource manager
Resource managers (RMs) manage data resources. Database systems are examples of resource managers. openUTM, however, also provides its own resource managers for accessing message queues, local memory areas and logging files, for instance. Applications access RMs via special resource manager interfaces. In the case of database systems, this will generally be SQL and in the case of openUTM RMs, it is the KDCS interface.
restart
See application restart, screen restart, service restart.

RFC1006
A protocol defined by the IETF (Internet Engineering Task Force) belonging to the TCP/IP family that implements the ISO transport services (transport class 0) based on TCP/IP.

RSA
Abbreviation for the inventors of the RSA encryption method (Rivest, Shamir and Adleman). This method uses a pair of keys that consists of a public key and a private key. A message is encrypted using the public key, and this message can only be decrypted using the private key. The pair of RSA keys is created by the UTM application.

SAT audit (BS2000/OSD)
Audit carried out by the SAT (Security Audit Trail) component of the BS2000 software product SECOS.

screen restart
If a dialog service is interrupted, openUTM again displays the dialog message of the last completed transaction on screen when the service restarts.

secondary storage area
Memory area in the KDCFILE secured by transaction logging and which can be accessed by the KDCS program unit with special calls. Local secondary storage areas (LSSBs) are assigned to one service. Global secondary storage areas (GSSBs) can be accessed by all services in a UTM application. Other secondary storage areas include the terminal-specific long-term storage (TLS) and the user-specific long-term storage (ULS).

selector
A selector identifies a service access point to services of one of the layers of the OSI reference model in the local system. Each selector is part of the address of the access point.

semaphore (Unix systems / Windows systems)
Unix systems and Windows systems resource used to control and synchronize processes.
Glossary

server
A server is an application which provides services. The computer on which the server applications are running is often also referred to as the server.

server-server communication
See distributed processing.

server side of a conversation (CPI-C)
This term has been superseded by acceptor.

service
Services process the jobs that are sent to a server application. A service of a UTM application comprises one or more transactions. The service is called with the service TAC. Services can be requested by clients or by other servers.

service access point
In the OSI reference model, a layer has access to the services of the layer below at the service access point. In the local system, the service access point is identified by a selector. During communication, the UTM application links up to a service access point. A connection is established between two service access points.

service chaining (KDCS)
When service chaining is used, a follow-on service is started without a dialog message specification after a dialog service has completed.

service-controlled queue
Message queue in which the calling and further processing of messages is controlled by services. A service must explicitly issue a KDCS call (DGET) to read the message. There are service-controlled queues in openUTM in the variants USER queue, TAC queue and temporary queue.

service restart (KDCS)
After a service has been terminated, 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.

service routine
See program unit.
service stacking (KDCS)
A terminal user can interrupt a running dialog service and insert a new dialog service. When the inserted service has completed, the interrupted service continues.

service TAC (KDCS)
Transaction code used to start a service.

session
Communication relationship between two addressable units in the network via the SNA protocol LU6.1.

session selector
The session selector identifies an access point in the local system to the services of the session layer of the OSI reference model.

shared code (BS2000/OSD)
Code which can be shared by several different processes.

shared memory
Virtual memory area which can be accessed by several different processes simultaneously.

shared objects (Unix systems / Windows systems)
Parts of the application program can be created as shared objects. These objects are linked to the application dynamically and can be replaced during live operation. Shared objects are defined with the KDCDEF statement SHARED-OBJECT.

sign-on check
See system access control.

sign-on service (KDCS)
Special dialog service in which program units control how a terminal user signs on to a client of a UTM application.

single-step service
Dialog service which encompasses precisely one dialog step.

single-step transaction
Transaction which encompasses precisely one dialog step.
Glossary

SOA
(Service-Oriented Architecture)
An SOA is a system architecture concept in which functions are implemented in the form of re-usable, technically independent, loosely coupled services. Services can be called independently of the underlying implementations via interfaces which may possess public and, consequently, trusted specifications. Service interaction is performed via a communication infrastructure made available for this purpose.

SOAP
SOAP (Simple Object Access Protocol) is a protocol used to exchange data between systems and run remote procedure calls. SOAP also makes use of the services provided by other standards, XML for the representation of the data and Internet transport and application layer protocols for message transfer.

socket connection
Transport system connection that uses the socket interface. The socket interface is a standard program interface for communication via TCP/IP.

standalone application
See standalone UTM application.

standalone UTM application
Traditional UTM application that is not part of a UTM cluster application.

standard primary working area (KDCS)
Area in main memory available to all KDCS program units. The contents of the area are either undefined or occupied with a fill character when the program unit starts execution.

start format
Format output to a terminal by openUTM when a user has successfully signed on to a UTM application (except after a service restart and during the sign-on service).

static configuration
Definition of the configuration during generation using the UTM tool KDCDEF.

SYSLOG file
See system log file.
synchronization point, consistency point
The end of a transaction. At this time, all the changes made to the application information during the transaction are saved to prevent loss in the event of a crash and are made visible to others. Any locks set during the transaction are released.

system access control
A check carried out by openUTM to determine whether a certain user ID is authorized to work with the UTM application. The authorization check is not carried out if the UTM application was generated without user IDs.

system log file
File or file generation to which openUTM logs all UTM messages for which SYSLOG has been defined as the message destination during execution of a UTM application.

TAC
See transaction code.

TAC queue
Message queue generated explicitly by means of a KDCDEF statement. A TAC queue is a service-controlled queue that can be addressed from any service using the generated name.

temporary queue
Message queue created dynamically by means of a program that can be deleted again by means of a program (see service-controlled queue).

terminal-specific long-term storage (KDCS)
Secondary storage area assigned to an LTERM, LPAP or OSI-PAP partner and which is retained after the application has terminated.

time-driven job
Job which is buffered by openUTM in a message queue up to a specific time until it is sent to the recipient. The recipient can be an asynchronous service of the same application, a TAC queue, a partner application, a terminal or a printer. Time-driven jobs can only be issued by KDCS program units.

timer process (Unix systems / Windows systems)
Process which accepts jobs for controlling the time at which work processes are executed. It does this by entering them in a job list and releasing them for processing after a time period defined in the job list has elapsed.
TNS (Unix systems / Windows systems)
Abbreviation for the Transport Name Service. TNS assigns a transport selector and a transport system to an application name. The application can be reached through the transport system.

Tomcat
see Apache Tomcat

transaction
Processing section within a service for which adherence to the ACID properties is guaranteed. If, during the course of a transaction, changes are made to the application information, they are either made consistently and in their entirety or not at all (all-or-nothing rule). The end of the transaction forms a synchronization point.

transaction code/TAC
Name which can be used to identify a program unit. The transaction code is assigned to the program unit during static or dynamic configuration. It is also possible to assign more than one transaction code to a program unit.

transaction rate
Number of transactions successfully executed per unit of time.

transfer syntax
With OSI TP, the data to be transferred between two computer systems is converted from the local format into transfer syntax. Transfer syntax describes the data in a neutral format which can be interpreted by all the partners involved. An Object Identifier must be assigned to each transfer syntax.

transport selector
The transport selector identifies a service access point to the transport layer of the OSI reference model in the local system.

transport system application
Application which is based directly on the transport system interface (e.g. CMX or socket). When transport system applications are connected, the partner type APPLI or SOCKET must be specified during configuration. A transport system application cannot be integrated in global transaction management.

TS application
See transport system application.
typed buffer (XATMI)
Buffer for exchanging typed and structured data between communication partners. Typed buffers ensure that the structure of the exchanged data is known to both partners implicitly.

UPIC
Carrier system for openUTM clients. UPIC stands for Universal Programming Interface for Communication.

UPIC client
The designation for openUTM clients with the UPIC carrier system.

user exit
This term has been superseded by event exit.

user ID
Identifier for a user defined in the configuration for the UTM application (with an optional password for system access control) and to whom special data access rights (system access control) have been assigned. A terminal user or client must specify this ID (and any password which has been assigned) when signing on to the UTM application. UTM applications can also be generated without user IDs.

user log file
File or file generation to which users write variable-length records with the LPUT call. The data from the KB header of the KDCS communication area is prefixed to every record. The user log file is subject to transaction management by openUTM.

USER queue
Message queue made available to every user ID by openUTM. A USER queue is a service-controlled queue and is always assigned to the relevant user ID. You can restrict the access of other UTM users to your own USER queue.

user-specific long-term storage
Secondary storage area assigned to a user ID, a session or an association and which is retained after the application has terminated.

USLOG file
See user log file.
Glossary

**UTM application**
A UTM application provides *services* which process jobs from *clients* or other applications. openUTM is responsible for transaction logging and for managing the communication and system resources. From a technical point of view, a UTM application is a process group which forms a logical server unit at runtime.

**UTM cluster application**
*UTM application* that has been generated for use on a cluster and that can be viewed logically as a *single* application.
In physical terms, a UTM cluster application is made up of several identically generated UTM applications running on the individual cluster *nodes*.

**UTM-controlled queue**
Message queues in which the calling and further processing of messages is entirely under the control of openUTM. See also *asynchronous job*, *background job* and *asynchronous message*.

**UTM-D**
See *openUTM-D*.

**UTM-F**
UTM applications can be generated as UTM-F applications (UTM fast). In the case of UTM-F applications, input from and output to hard disk is avoided in order to increase performance. This affects input and output which *UTM-S* uses to save user data and transaction data. Only changes to the administration data are saved.

**UTM message**
Messages are issued to *UTM message destinations* by the openUTM transaction monitor or by UTM tools (such as *KDCDEF*). A message comprises a message number and a message text, which can contain *inserts* with current values. Depending on the message destination, either the entire message is output or only certain parts of the message, such as the inserts).

**UTM page**
A UTM page is a unit of storage with a size of either 2Kb or 4Kb. The size is specified when the *UTM application* is generated. The page pool and the restart area for the KDCFILE are divided into units of the size of a UTM page.
utmpath (Unix systems / Windows systems)
The directory under which the openUTM components are installed is referred to as utmpath in this manual. To ensure that openUTM runs correctly, the environment variable UTMPATH must be set to the value of utmpath. On Unix systems, you must set UTMPATH before a UTM application is started. On Windows systems, UTMPATH is set on installation.

UTM-S
In the case of UTM-S applications, openUTM saves all user data as well as the administration data beyond the end of an application and any system crash which may occur. In addition, UTM-S guarantees the security and consistency of the application data in the event of any malfunction. UTM applications are usually generated as UTM-S applications (UTM secure).

UTM SAT administration (BS2000/OSD)
UTM-SAT administration functions control which UTM events relevant to security which occur during operation of a UTM application are to be logged by SAT. Special authorization is required for UTM-SAT administration.

UTM terminal
This term has been superseded by LTERM partner.

virtual connection
Assignment of two communication partners.

warm start
Start of a UTM application after it has terminated abnormally (see abnormal termination). During a warm start, the KDCFILE is restored to a consistent state.

Web service
Application which runs on a Web server and is (publicly) available via a standardized, programmable interface. Web services technology makes it possible to make UTM program units available for modern Web client applications independently of the programming language in which they were developed.

work process (Unix systems / Windows systems)
A process within which the services of a UTM application run.

WS4UTM
WS4UTM (WebServices for openUTM) provides you with a convenient way of making a service of a UTM application available as a Web service.
**XATMI**

XATMI (X/Open Application Transaction Manager Interface) is a program interface standardized by X/Open for program-program communication in open networks. The XATMI interface implemented in openUTM complies with X/Open’s XATMI CAE Specification. The interface is available in COBOL and C. In openUTM, XATMI can communicate via the OSI TP, \( LU6.1 \) and UPIC protocols.

**XHCS (BS2000/OSD)**

XHCS (Extended Host Code Support) is a BS2000/OSD software product providing support for international character sets.

**XML**

XML (eXtensible Markup Language) is a metalanguage standardized by the W3C (WWW Consortium) in which the interchange formats for data and the associated information can be defined.
## Abbreviations

Please note: Some of the abbreviations used here derive from the German acronyms used in the original German product(s).

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACSE</td>
<td>Association Control Service Element</td>
</tr>
<tr>
<td>AEQ</td>
<td>Application Entity Qualifier</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>AET</td>
<td>Application Entity Title</td>
</tr>
<tr>
<td>APT</td>
<td>Application Process Title</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ASE</td>
<td>Application Service Element</td>
</tr>
<tr>
<td>Axis</td>
<td>Apache eXtensible Interaction System</td>
</tr>
<tr>
<td>BCAM</td>
<td>Basic Communication Access Method</td>
</tr>
<tr>
<td>BER</td>
<td>Basic Encoding Rules</td>
</tr>
<tr>
<td>BLS</td>
<td>Binder - Loader - Starter (BS2000/OSD)</td>
</tr>
<tr>
<td>CCP</td>
<td>Communication Control Program</td>
</tr>
<tr>
<td>CCR</td>
<td>Commitment, Concurrency and Recovery</td>
</tr>
<tr>
<td>CCS</td>
<td>Coded Character Set</td>
</tr>
<tr>
<td>CCSN</td>
<td>Coded Character Set Name</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CID</td>
<td>Control Identification</td>
</tr>
<tr>
<td>CMX</td>
<td>Communication Manager in Unix Systems</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CPI-C</td>
<td>Common Programming Interface for Communication</td>
</tr>
<tr>
<td>CRM</td>
<td>Communication Resource Manager</td>
</tr>
<tr>
<td>CRTE</td>
<td>Common Runtime Environment (BS2000/OSD)</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DC</td>
<td>Data Communication</td>
</tr>
<tr>
<td>DCAM</td>
<td>Data Communication Access Method</td>
</tr>
</tbody>
</table>
**Abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Mode</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DLM</td>
<td>Distributed Lock Manager (BS2000/OSD)</td>
</tr>
<tr>
<td>DMS</td>
<td>Data Management System</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name Service</td>
</tr>
<tr>
<td>DP</td>
<td>Distributed Processing</td>
</tr>
<tr>
<td>DSS</td>
<td>Terminal (Datensichtstation)</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>DTP</td>
<td>Distributed Transaction Processing</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>Extended Binary-Coded Decimal Interchange Code</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans$^{TM}$</td>
</tr>
<tr>
<td>FGG</td>
<td>File Generation Group</td>
</tr>
<tr>
<td>FHS</td>
<td>Format Handling System</td>
</tr>
<tr>
<td>FT</td>
<td>File Transfer</td>
</tr>
<tr>
<td>GSSB</td>
<td>Global Secondary Storage Area</td>
</tr>
<tr>
<td>HIPLEX$^{®}$</td>
<td>Highly Integrated System Complex (BS2000/OSD)</td>
</tr>
<tr>
<td>HLL</td>
<td>High-Level Language</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IFG</td>
<td>Interactive Format Generator</td>
</tr>
<tr>
<td>ILCS</td>
<td>Inter-Language Communication Services (BS2000/OSD)</td>
</tr>
<tr>
<td>IMS</td>
<td>Information Management System (IBM)</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-Process Communication</td>
</tr>
<tr>
<td>IRV</td>
<td>International Reference Version</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition Technologie</td>
</tr>
<tr>
<td>JCA</td>
<td>Java Connector Architecture</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JEE5</td>
<td>Java Enterprise Edition 5.0</td>
</tr>
<tr>
<td>KAA</td>
<td>KDCS Application Area</td>
</tr>
<tr>
<td>KB</td>
<td>Communication Area</td>
</tr>
<tr>
<td>KBPRG</td>
<td>KB Program Area</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>KDCS</td>
<td>Compatible Data Communication Interface</td>
</tr>
<tr>
<td>KTA</td>
<td>KDCS Task Area</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LCF</td>
<td>Local Configuration File</td>
</tr>
<tr>
<td>LLM</td>
<td>Link and Load Module (BS2000/OSD)</td>
</tr>
<tr>
<td>LSSB</td>
<td>Local Secondary Storage Area</td>
</tr>
<tr>
<td>LU</td>
<td>Logical Unit</td>
</tr>
<tr>
<td>MIGRAT</td>
<td>Migration Program</td>
</tr>
<tr>
<td>MQ</td>
<td>Message Queuing</td>
</tr>
<tr>
<td>MSCF</td>
<td>Multiple System Control Facility (BS2000/OSD)</td>
</tr>
<tr>
<td>NB</td>
<td>Message Area</td>
</tr>
<tr>
<td>NEA</td>
<td>Network Architecture for TRANSDATA Systems</td>
</tr>
<tr>
<td>NFS</td>
<td>Network File System/Service</td>
</tr>
<tr>
<td>NLS</td>
<td>Native Language Support</td>
</tr>
<tr>
<td>OCMX</td>
<td>OLE Control Extension</td>
</tr>
<tr>
<td>OLTP</td>
<td>Online Transaction Processing</td>
</tr>
<tr>
<td>OML</td>
<td>Object Module Library</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>OSI TP</td>
<td>Open System Interconnection Transaction Processing</td>
</tr>
<tr>
<td>OSS</td>
<td>OSI Session Service</td>
</tr>
<tr>
<td>PCMX</td>
<td>Portable Communication Manager</td>
</tr>
<tr>
<td>PDN</td>
<td>Program System for Remote Data Processing and Network Control</td>
</tr>
<tr>
<td>PID</td>
<td>Process Identification</td>
</tr>
<tr>
<td>PIN</td>
<td>Personal Identification Number</td>
</tr>
<tr>
<td>PLU</td>
<td>Primary Logical Unit</td>
</tr>
<tr>
<td>RAV</td>
<td>Computer Center Accounting Procedure</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Definition File</td>
</tr>
<tr>
<td>RM</td>
<td>Resource Manager</td>
</tr>
<tr>
<td>RSA</td>
<td>Encryption algorithm according to Rivest, Shamir, Adleman</td>
</tr>
<tr>
<td>RSO</td>
<td>Remote SPOOL Output (BS2000/OSD)</td>
</tr>
<tr>
<td>RTS</td>
<td>Runtime System</td>
</tr>
<tr>
<td>SAT</td>
<td>Security Audit Trail (BS2000/OSD)</td>
</tr>
<tr>
<td>SECOS</td>
<td>Security Control System</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>SLU</td>
<td>Secondary Logical Unit</td>
</tr>
<tr>
<td>SM2</td>
<td>Software Monitor 2 (BS2000/OSD)</td>
</tr>
<tr>
<td>SNA</td>
<td>Systems Network Architecture</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPAB</td>
<td>Standard Primary Working Area</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSB</td>
<td>Secondary Storage Area</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign-On</td>
</tr>
<tr>
<td>TAC</td>
<td>Transaction Code</td>
</tr>
<tr>
<td>TCEP</td>
<td>Transport Connection End Point</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transport Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>TIAM</td>
<td>Terminal Interactive Access Method</td>
</tr>
<tr>
<td>TLS</td>
<td>Terminal-Specific Long-Term Storage</td>
</tr>
<tr>
<td>TM</td>
<td>Transaction Manager</td>
</tr>
<tr>
<td>TNS</td>
<td>Transport Name Service</td>
</tr>
<tr>
<td>TP</td>
<td>Transaction Processing (Transaction Mode)</td>
</tr>
<tr>
<td>TPR</td>
<td>Privileged Function State in BS2000/OSD (Task Privileged)</td>
</tr>
<tr>
<td>TPSU</td>
<td>Transaction Protocol Service User</td>
</tr>
<tr>
<td>TSAP</td>
<td>Transport Service Access Point</td>
</tr>
<tr>
<td>TSN</td>
<td>Task Sequence Number</td>
</tr>
<tr>
<td>TU</td>
<td>Non-Privileged Function State in BS2000/OSD (Task User)</td>
</tr>
<tr>
<td>TX</td>
<td>Transaction Demarcation (X/Open)</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UDS</td>
<td>Universal Database System</td>
</tr>
<tr>
<td>UDT</td>
<td>Unstructured Data Transfer</td>
</tr>
<tr>
<td>ULS</td>
<td>User-Specific Long-Term Storage</td>
</tr>
<tr>
<td>UPIC</td>
<td>Universal Programming Interface for Communication</td>
</tr>
<tr>
<td>USP</td>
<td>UTM Socket Protocol</td>
</tr>
<tr>
<td>UTM</td>
<td>Universal Transaction Monitor</td>
</tr>
<tr>
<td>UTM-D</td>
<td>UTM Variant for Distributed Processing in BS2000</td>
</tr>
<tr>
<td>UTM-F</td>
<td>UTM Fast Variant</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>UTM-S</td>
<td>UTM Secure Variant</td>
</tr>
<tr>
<td>UTM-XML</td>
<td>openUTM XML Interface</td>
</tr>
<tr>
<td>VGID</td>
<td>Service ID</td>
</tr>
<tr>
<td>VTSU</td>
<td>Virtual Terminal Support</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WS4UTM</td>
<td>Web-Services for openUTM</td>
</tr>
<tr>
<td>WSDD</td>
<td>Web Service Deployment Descriptor</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XA</td>
<td>X/Open Access Interface</td>
</tr>
<tr>
<td></td>
<td>(X/Open interface for access to the resource manager)</td>
</tr>
<tr>
<td>XAP</td>
<td>X/OPEN ACSE/Presentation programming interface</td>
</tr>
<tr>
<td>XAP-TP</td>
<td>X/OPEN ACSE/Presentation programming interface</td>
</tr>
<tr>
<td></td>
<td>Transaction Processing extension</td>
</tr>
<tr>
<td>XATMI</td>
<td>X/Open Application Transaction Manager Interface</td>
</tr>
<tr>
<td>XCS</td>
<td>Cross Coupled System</td>
</tr>
<tr>
<td>XHCS</td>
<td>eXtended Host Code Support</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Related publications

PDF files of all openUTM manuals are included on the Enterprise DVD with open platforms and on the openUTM WinAdmin DVD (for BS2000/OSD).

All manuals are available as online manuals, see http://manuals.ts.fujitsu.com.

openUTM documentation

openUTM
Concepts and Functions
User Guide

openUTM
Programming Applications with KDCS for COBOL, C and C++
Core Manual

openUTM
Generating Applications
User Guide

openUTM
Using openUTM Applications under BS2000/OSD
User Guide

openUTM
Using openUTM Applications under Unix Systems and Windows Systems
User Guide

openUTM
Administering Applications
User Guide

openUTM
Messages, Debugging and Diagnostics in BS2000/OSD
User Guide
Related publications

openUTM
Messages, Debugging and Diagnostics in Unix Systems and Windows Systems
User Guide

openUTM (BS2000/OSD, Unix systems, Windows NT)
Creating Applications with X/Open Interfaces
Core Manual

openUTM
XML for openUTM

openUTM Client (Unix systems)
for the OpenCPIC Carrier System
Client-Server Communication with openUTM
User Guide

openUTM WinAdmin
Graphical Administration Workstation for openUTM
Online description and online help system

openUTM, openUTM-LU62
Distributed Transaction Processing
between openUTM and CICS, IMS and LU6.2 Applications
User Guide

openUTM (BS2000/OSD)
Programming Applications with KDCS for Assembler
Supplement to Core Manual

openUTM (BS2000/OSD)
Programming Applications with KDCS for Fortran
Supplement to Core Manual

openUTM (BS2000/OSD)
Programming Applications with KDCS for Pascal-XT
Supplement to Core Manual

openUTM (BS2000/OSD)
Programming Applications with KDCS for PL/I
Supplement to Core Manual
Related publications

**WS4UTM** (Unix systems and Windows systems)
*WebServices for openUTM*

*openUTM*
*Master Index*

**Documentation for the openSEAS product environment**

**BeanConnect**
*User Guide*

**JConnect**
*Connecting Java Clients to openUTM*
*User documentation and Java docs*

**WebTransactions**
*Concepts and Functions*

**WebTransactions**
*Template Language*

**WebTransactions**
*Web Access to openUTM Applications via UPIC*

**WebTransactions**
*Web Access to MVS Applications*

**WebTransactions**
*Web Access to OSD Applications*
Related publications

Documentation for the BS2000/OSD environment

Most of these manuals are available in printed form which must be paid and ordered separately at http://manualshop.ts.fujitsu.com

**AID (BS2000/OSD)**
Advanced Interactive Debugger
Core Manual
User Guide

**BCAM (BS2000/OSD)**
BCAM Volume 1/2
User Guide

**BINDER (BS2000/OSD)**
User Guide

**BS2000/OSD**
Executive Macros
User Guide

**BS2000/OSD-BC**
BLSSERV
Dynamic Binder Loader / Starter
User Guide

**DCAM (BS2000/OSD)**
COBOL Calls
User Guide

**DCAM (BS2000/OSD)**
Macros
User Guide

**DCAM (BS2000/OSD)**
Program Interfaces
Description

**FHS (BS2000/OSD)**
Format Handling System for openUTM, TIAM, DCAM
User Guide

**IFG for FHS**
User Guide
FHS-DOORS (BS2000/OSD, MS-Windows)
Graphical Interface for BS2000/OSD Applications
User Guide

HIPLEX AF (BS2000/OSD)
High-Availability of Applications in BS2000/OSD
Product Manual

HIPLEX MSCF (BS2000/OSD)
BS2000 Processor Networks
User Guide

IMON (BS2000/OSD)
Installation Monitor
User Guide

MT9750 (MS Windows)
9750 Emulation under Windows
Product Manual

OMNIS/OMNIS-MENU (BS2000/OSD)
Functions and Commands
User Guide

OMNIS/OMNIS-MENU (BS2000)
Administration and Programming
User Guide

OMNIS-MENU (BS2000/OSD)
User Guide

OSS (BS2000/OSD)
OSI Session Service
User Guide

RSO (BS2000/OSD)
Remote SPOOL Output
User Guide

SECOS (BS2000/OSD)
Security Control System
User Guide
Related publications

SECOS (BS2000/OSD)
Security Control System
Ready Reference

SESAM/SQL (BS2000/OSD)
Database Operation
User Guide

openSM2 (BS2000/OSD)
Software Monitor
Volume 1: Administration and Operation

TIAM (BS2000/OSD)
User Guide

UDS/SQL (BS2000/OSD)
Database Operation
User Guide

Unicode in BS2000/OSD
Introduction

VTSU (BS2000/OSD)
Virtual Terminal Support
User Guide

XHCS (BS2000/OSD)
8-Bit Code and Unicode Support in BS2000/OSD
User Guide
Documentation for the Unix system environment

**CMX V6.0** (Solaris)
*Operation and Administration*
User Guide

**CMX V6.0** (Unix systems)
*Operation and Administration*
User Guide

**CMX V6.0**
Programming CMX Applications
Programming Guide

**OSS (SINIX)**
*OSI Session Service*
User Guide

**PRIMECLUSTER™**
*Concepts Guide (Solaris, Linux)*

**openSM2**
The documentation of openSM2 (Solaris) and openSM2 (Linux) is provided in the form of detailed online help systems, which are delivered with the product.
Related publications

Other publications

**CPI-C (X/Open)**
Distributed Transaction Processing
X/Open CAE Specification, Version 2
ISBN 1 85912 135 7

**Reference Model Version 2 (X/Open)**
Distributed Transaction Processing
X/Open Guide
ISBN 1 85912 019 9

**TX (Transaction Demarcation) (X/Open)**
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 094 6

**XTAMI (X/Open)**
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 130 6

**XML**
W3C specification (www consortium)
Web page: [http://www.w3.org/XML](http://www.w3.org/XML)
Index

#format 101
*format 101
+format 101
.DEFAULT 331, 338

19Z, MGET 239

A
access authorization check
   without password 227
access control
   UTM 304
access list concept 114
activate
   UPIC trace 365
active RSA key 121
Active Server page 37
ActiveX Control Upic.ocx 27
additional function of the UPIC carrier system 213
addressing formats 319
addressing information 320
addressing information for the network 321
administration journal 399
advanced functions 246
AES key 120
alignment 281
Allocate 136
ANSI 307, 349, 383
ANSI compiler 383
apartment model 40
application
   port for local application 265

applifile 180
ASCII 320
ASCII character 106
ASCII conversion 283
ASCII/EBCDIC conversion 104
ASN.1 type 282
asynchronous request/response paradigm 279
attribute fields
   format 101
automatic conversion 105
automation server 68

B
BADTAC 358
BCMAP 322
big endian 281
blocking
   Receive 185
   receive, overview 96
   Receive, timer 255
   Receive_Mapped_Data 196
   Receive_Mapped_Data, timer 255
   set 258
BlockingReceive (UpicB.ocx) 45
BOOL Peek 82
BS2000
   BCMAP 322
calling XATMI 301
code conversion using upicfile 325
compiling 357
ftp 362
incoming code conversion 139
job variable 348
linking 357
logging file 364
Index

BS2000 (cont.)
  outgoing code conversion 140
  ping 361
  runtime environment 358
  telnet 362
  UPIC trace 365
  upicfile 323
buffer
  define 299
  for data 93
  maximum size 294
  Receive 185
  Receive_Mapped_Data 197
  secondary information 167
  send data 212
  Send_Mapped_Data 210
BUFFER statement 299
C
  C data types 281
  C++ class CUpic 69
call
  Enable_UTM_UPIC 147
  Extract_Conversation_State 159
  Extract_Cursor_Offset 163
  Extract_Partner_LU_Name 165
  Extract_Secondary_Information 167
  Send_Mapped_Data 210
  Set_Allocate_Timer 213
  Set_Conversation_Encryption_Level 220
  Set_Conversation_New_Password 224
  Set_Conversation_Security_Password 227
  Set_Partner_Host_Name 241
  Set_Partner_IP_Address 243
  Set_Partner_LU_Name 246
  Set_Partner_Port 249
  Set_Partner_Tsel 251
  Set_Partner_Tsel_Format 253
  Set_Receive_Timer 255
  Specify_Local_Port 265
  Specify_Local_Tsel 267
  Specify_Local_Tsel_Format 269
calls
  UPIC functions 135
carrier system
  connect (XATMI) 285
  carrier system OpenCPIC 27
  carrier system UPIC 27
  CCMSDT 236
  CD 333
change
  from Send to Receive state 182
  send authorization 94
character set conversion 283
character set encoding 283
CHARACTER_CONVERSION 89, 105
set 234
characteristics (XATMI) 294
check TCP/IP connection 361
class ID 40
ClassCUpic 76
client 25
  initialize 286
  XATMI 275
client context 119
  query 151
client/server concept 25
CLIENT_CONTEXT 89
cluster 33
cluster administration journal 399
cluster application 33
CM_COMMUNICATION_PROTOCOL_34
  query 154
  set in program 218
CM_COMMUNICATION_PROTOCOL_40
  query 154
  set in program 218
CM_DEALLOATED_ABEND 359
  PEND ER/FR 361
CM_RECEIVE_AND_WAIT 185, 258
CM_RECEIVE_IMMEDIATE 185, 259
CM_SECURITY_PWD_EXPIRED_RETRY 116
CMALLC 136
CMCNVI 139
CMCNVO 140
CMCOBOL 273
CMDEAL 141
CMDFDE 143
CMDISA 145  
CMECEL 156  
CMECO 163  
CMECP 154  
CMECS 159  
CMENAB 147  
CMEPLN 165  
CMESI 167  
CMESRC 170  
CMINIT 178  
CMPTR 182  
CMRCV 185  
CMRCVM 196  
CMSAT 213  
CMSCC 215  
CMSCEL 220  
CMSCP 218  
CMSCSN 224  
CMSCSP 227  
CMSCST 230  
CMSCSU 232  
CMSEND 207  
CMSFK 238  
CMSLP 265  
CMSLT 267  
CMSLTF 269  
CMSNDM 210  
CMSPHN 241  
CMSPIA 243  
CMSPLN 246  
CMSPP 249  
CMSPT 251  
CMSPTF 253  
CMRRCCT 255  
CMSRT 258  
CMSSL 261  
CMSSRC 271  
CMSTPN 263  
CMX 31, 322, 354  
COBOL, CPI-C interface 273  
  code  
    data types 282, 299  
    node application 333  
  code conversion 104  
  code conversion for Windows 106  
  COM 35  
  COM/DCOM interface 35  
  comment line  
    LCF 295  
    upicfile 323  
  common data types 281  
  communication  
    with a UTM server 107  
  communication paradigm 276  
    asynchronous 279  
    conversational 280  
    synchronous 278  
  communication protocol  
    specify in upicfile 328  
  Communication Resource Manager 275  
  communication system 32  
  COMMUNICATION_PROTOCOL 89  
  openUTM 92  
  COMP statement 273  
  compatible  
    CPI-C program 262, 383  
  compile  
    CPI-C program (Unix system) 354  
    CPI-C program (Windows system) 349  
    under BS2000/OSD 357  
    XATMI program 307  
  compiler option 383  
  UTM_ON_WIN32 349, 383  
  complete message, receive 203  
  configuration of the UPIC carrier system 317  
  configuration with TNS entries 322  
  configuration without upicfile 318  
  configure  
    UPIC local with Visual C++ 352  
    XATMI 295  
  connect carrier system (XATMI) 285  
  connection setup  
    problems 361  
    with encryption 121  
  connection shutdown  
    during receive 199  
    during sending 211  
    when changing Receive state 183
connection via UPIC-Local 213
CONNECT-MODE 342
continuation character 295
conversation 280
  CPI-C 86
    establish 136
    establish (default values) 178
    parallel 125
    query state of 159
    terminate 141, 143
Conversation Characteristic 86
  conversation_type 87
  deallocate_type 88
  ENCRYPTION-LEVEL 89
  HOSTNAME 89
  IP-ADDRESS 89
  modifiable 88
  partner_LU_name 88
  partner_LU_name_length 88
  PORT 89
  receive_type 88, 185
  receive_type (Receive) 185
  receive_type (Receive_Mapped_Data) 196
  receive_type (Set.Receive_Type) 258
  return_control 87
  RSA-KEY 89
  security_new_password 88
  security_new_password_length 88
  security_password 88, 227
  security_password_length 88, 227
  security_type 88
  security_user_ID 88, 230
  security_user_ID_length 88
  send_type 87
  status_received (Send_Data) 207, 210
  sync_level 87, 261
  TP_name 88
  TP_name_length 88
  T-SEL 89
  T-SEL-FORMAT 89
  UPIC specific 89
conversation state 86
conversation_ID 179
Initialize_Conversation 179
  length 132
  conversation_type 87
  conversational paradigm 280
  configure 298
conversion 104, 283
  ASCII to EBCDIC 140
  ASCII to EBCDIC (Windows) 106
  automatic 324
  EBCDIC to ASCII 139
  EBCDIC to ASCII (Windows) 106
  Euro symbol 105
conversion table 106
  edit 105
  Windows 139, 140
Convert_Incoming 139
  overview 104
Convert_Outgoing 140
  overview 104
ConvertIncoming (UpicB.ocx) 59
ConvertOutgoing (UpicB.ocx) 60
COPY element
  COBOL 273
copy-cobol85 273
core 360
CPI-C 13
  concealment 26
  version 2 382
CPI-C application 28
  implementation 347
  sequence of calls 91
  structure 91
CPI-C calls 132
  C 132
  COBOL 273
  sequence 91
CPI-C interface 132
CPI-C program 28
  access to services 29
  link (Unix system) 354
  portability 262
  sign off 145
  sign on 147
  start 349, 354
CPI-C terms, definition 86
create LCF 300
C-source 106
CUPIC class 26
CUPIC object 26
CUpic security 75
CUPIC wrapper class 26
CUpicLocAddr 69, 71
CUpicRemAddr 69, 73
cursor position 104
  offset query 163
D
data
  receive 185, 192, 203
  receive in full 192
  receive with format identifier 196
  send 207
  send with format identifier 210
data access control functions 114
data buffer, XATMI 281
data received
  display(Receive_Mapped_Data) 197
data security 115
data structure
  name 299
data types
  XATMI 281
data volumes
  large 109, 294
data_received
  Receive_Mapped_Data 197
database accesses 94
DataReceived (UpicB.ocx) 61
DCOM 35
deactivating
  UPIC trace 369
Deallocate 141
deallocate_type 88
  set 236
Def file with EXPORT statements 106
default entries 70
DEFAULT name 147
  of a client 130
DEFAULT server 130
  defining 338
DEFAULT service 130
  defining 338
default service
  define 331
default values
  GLOBAL NAME 343
Deferred_Deallocate 143
  call 143
DES key 120
destination-name 297
determine conversation ID 179
diagnostic documentation 363
diagnostics
  PCMX 370
  UPIC carrier system 363
directory
  trace files 366
Disable_UTM_UPIC 145
Disconnect (UpicB.ocx) 55
dispatch interface 40
Distributed Transaction Processing 94, 111
documentation
  summary 15
dynamic library 106
E
EBCDIC 320
EBCDIC code 106
EBCDIC conversion
  XATMI 283
EBCDIC to ASCII
  conversion 104
edit conversion table 105
edit UPIC trace 369
empty format identifier 203
empty local name 341
empty Symbolic Destination Name 319
Enable_UTM_UPIC 147
  local name 341
encryption 76, 120
equivalent at TAC 123
Index

encryption level 121
  associated key pairs 120
  modify 157
  query 156
  set 220
encryption methods 120
encryption_level
  too high 221
ENCRYPTION-LEVEL 89
end of line character
  upicfile 337
end service 275
endless loop 96
end-of-line character, upicfile 330
environment variable
  CPI-C 347
    set for CPI-C (Windows) 350
    UPIC local (Unix system) 356
error condition in the CPI-C program 146
error diagnosis
  CPI-C 360
  XATMI 309
error handling
  XATMI 291
error message (CPI-C) 360
error situation
  CPI-C 359
establish
  conversation 136
  conversation (default values) 178
Euro symbol, conversion 105
event
  DataReceived (UpicB.ocx) 61
  XATMI 291
example
  multiple conversations 128
  OCXTestContainer (Windows) 375
  side information entry 373
  TNS entry 373
  UpicAutClt (Windows) 375
  UpicAutSvr (Windows) 375
  UpicB.ocx 64
  upmulti (Windows 32-bit) 374
  uptac (Windows) 373
  utp (Windows) 373
examples for Windows 371
exchanging formats 98
exchanging messages 92
  functions for 92
exit 146
expiry of timer
  Receive 188
Extract_Client_Context 119, 151
Extract_Communication_Protocol 154
Extract_Conversation_Encryption_Level call 156
Extract_Conversation_State call 159
Extract_Conversion call 161
Extract_Cursor_Offset call 163
Extract_Partner_LU_Name 165
Extract_Secondary_Information 167
Extract_Secondary_Information call 167
Extract_Secondary_Return_Code 114, 170
Extract_Transaction_State 175
F
F keys
  overview 102
  set 238
file descriptor 351, 355
follow-up states of a conversation 86
-format 101
format data
  exchange 98
FORMAT event exit 101
format identifier
  empty 203
  receive 196
  send 210
format name 98, 101
format of the names 320
format types 101
formats
  receive 98
  send 98
FPUT 358
ftp 361
function calls 78
function declaration
  Extract_Partner_LU_Name 166
function keys 102
  activate 102
function_key
  Set_Function_Key 238
FunctionKey (UpicB.ocx) 43
functions
  message exchange 92
G
  generation of the UTM partner application 317
GetLastError (UpicB.ocx) 58
global communication 318
globally unique identifier 40
Grace-Sign-On
  password expired 116
GUID 40
H
  HD 324
header file
  CPI-C 349
  CPI-C (Unix system) 354
helper classes 69, 71
host name
  partner application 241
HOSTNAME 89
HP-UX 14
I
  identifier
    conversation 179
    conversion 324
IncompleteDataReceived (UpicB.ocx) 57
InConversation (UpicB.ocx) 55
initialization parameters
  XATMI 304
initialization value
  conversation characteristics 88, 89
initialize
  XATMI client 286
initialize state 86
Initialize_Conversation 178
INPUT exit 358
input parameters
  CPI-C 132
InSendingState (UpicB.ocx) 56
installation of openUTM client 38
int Call 80
int Rcv 78
int RcvMulti 79
int Restart 81
int SetTselFormat 72, 74
int Snd 78
int SndLast 78
int SndRcv 80
interfaces
  overview 26
intermediate service 275
internal error 360
internal-service-name 297
interprocess communication 319
invalid 187
  password 137, 187, 198, 228, 233
  user ID 198
  UTM user ID 187
IP address
  partner application 243
IP-ADDRESS 89
IPv4 327, 335
IPv6 245, 327, 335
J
  Java 37
job variable 348
  setting (example) 358
    XATMI 309
K
  K keys
    overview 102
    set 238
  K&R compiler 383
  KCMF 212
  kcsaeaa.c 106
  kcxaent.c 106
Index

KDCDISP 90, 117
  error 199
KDCRECVR (XATMI) 277, 297
Kerberos
  return code 171
  return code (Receive) 189
  return code (Receive_Mapped_Data) 200
key code 114
keywords 90

L
large data volumes 109, 294
last output message
  CPI-C 118
  XATMI 117, 277, 297
LC definition file 300
LC description files 300
LCF 276, 295, 300
  create 300
length
  data (Send_Mapped_Data) 210
  format identifier 196
  partner name in upicfile 325, 333
  symbolic destination name 325, 333
link
  CPI-C program 349
  CPI-C program (Unix system) 354
  UPIC local (Unix system) 356
  UPIC local (Windows) 351
  XATMI program 307
linking
  BS2000 357
Linux distribution 14
little endian 281
LN 339
LN.DEFAULT entry 147
local application
  set port 265
    set transport selector 267
      set transport selector format 269
local application name 90
local client name 286
local configuration
  code for syntax 282
  local configuration definition file 295
  local configuration file 276, 300
    create 300
    define name 302
  local connection 30
  local definition file
    for XATMI sample application 374
  local name 70, 90, 323
    empty 341
    Enable_UTM_UPIC 147, 341
    upicfile 305, 339
  local port number 320
  local send buffer 94
  local_name 90
    Enable_UTM_UPIC 147
    LocalName (UpicB.ocx) 43
lock code 114
log file
  BS2000 364
  UPIC 363
long-term storage area 115
LTERM pool
  for multiple sign-on 131

M
machine dependencies 281
main thread 125
map_name
  overview 99
  Receive_Mapped_Data 196
  Send_Mapped_Data 210
map_name_length
  Receive_Mapped_Data 196
  Send_Mapped_Data 210
MapName (UpicB.ocx) 45
MapOn (UpicB.ocx) 45
mathematical library 307
mathlib 307
maximum lengths 294
maximum message length 294
message exchange 92
message length
  maximum 294
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message segment</td>
<td>95</td>
</tr>
<tr>
<td>Receive</td>
<td>95</td>
</tr>
<tr>
<td>Messages</td>
<td>95</td>
</tr>
<tr>
<td>Send</td>
<td>93</td>
</tr>
<tr>
<td>Xatmigen</td>
<td>314</td>
</tr>
<tr>
<td>Metasyntax</td>
<td>23</td>
</tr>
<tr>
<td>Methods of UpicB.ocx</td>
<td>48</td>
</tr>
<tr>
<td>MGET</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>19Z</td>
</tr>
<tr>
<td></td>
<td>239</td>
</tr>
<tr>
<td>MGET NT</td>
<td>99</td>
</tr>
<tr>
<td>Microsoft Office</td>
<td>35</td>
</tr>
<tr>
<td>Microsoft Visual C++ Developer Studio</td>
<td>106</td>
</tr>
<tr>
<td>Migration</td>
<td></td>
</tr>
<tr>
<td>CPI-C version 2</td>
<td>382</td>
</tr>
<tr>
<td>MODE</td>
<td></td>
</tr>
<tr>
<td>Service paradigm</td>
<td>298</td>
</tr>
<tr>
<td>MPUT</td>
<td>109</td>
</tr>
<tr>
<td>MPUT NT</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>97</td>
</tr>
<tr>
<td>MSCF</td>
<td>401</td>
</tr>
<tr>
<td>Multiple clients</td>
<td>115</td>
</tr>
<tr>
<td>CPI-C program runs</td>
<td>149</td>
</tr>
<tr>
<td>Send and receive</td>
<td>109</td>
</tr>
<tr>
<td>Multiple conversations</td>
<td>125</td>
</tr>
<tr>
<td>Example</td>
<td>128</td>
</tr>
<tr>
<td>Multiple sign-on</td>
<td></td>
</tr>
<tr>
<td>LTERM pool for</td>
<td>131</td>
</tr>
<tr>
<td>To UTM</td>
<td>149</td>
</tr>
<tr>
<td>With same name</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>342</td>
</tr>
<tr>
<td>Multi-step service</td>
<td>97</td>
</tr>
<tr>
<td>Multithreading</td>
<td>125</td>
</tr>
<tr>
<td>Multithreading-capable</td>
<td>32</td>
</tr>
<tr>
<td>Node application</td>
<td>33</td>
</tr>
<tr>
<td>Non-blocking</td>
<td></td>
</tr>
<tr>
<td>Receive</td>
<td>185</td>
</tr>
<tr>
<td>Receive, overview</td>
<td>97</td>
</tr>
<tr>
<td>Receive_Mapped_Data</td>
<td>196</td>
</tr>
<tr>
<td>Set</td>
<td>258</td>
</tr>
<tr>
<td>Notational conventions</td>
<td>22</td>
</tr>
<tr>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OCTET STRING</td>
<td>282</td>
</tr>
<tr>
<td>OCXTestContainer</td>
<td></td>
</tr>
<tr>
<td>Sample program for Windows</td>
<td>375</td>
</tr>
<tr>
<td>Office</td>
<td>35</td>
</tr>
<tr>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>Cursor position</td>
<td>163</td>
</tr>
<tr>
<td>The cursor position in the format</td>
<td>104</td>
</tr>
<tr>
<td>OLE</td>
<td>35</td>
</tr>
<tr>
<td>OpenCPIC carrier system</td>
<td>27</td>
</tr>
<tr>
<td>OpenUTM</td>
<td></td>
</tr>
<tr>
<td>Encryption of user data</td>
<td>120</td>
</tr>
<tr>
<td>Format identifier</td>
<td>101</td>
</tr>
<tr>
<td>Function keys</td>
<td>102</td>
</tr>
<tr>
<td>User concept</td>
<td>113</td>
</tr>
<tr>
<td>Version</td>
<td>92</td>
</tr>
<tr>
<td>OpenUTM cluster</td>
<td></td>
</tr>
<tr>
<td>Rules for upicfile</td>
<td>332</td>
</tr>
<tr>
<td>Symbolic destination name</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>332</td>
</tr>
<tr>
<td>OpenUTM cluster application</td>
<td>21</td>
</tr>
<tr>
<td>OpenUTM-Client interface</td>
<td>69</td>
</tr>
<tr>
<td>Operating system</td>
<td>32</td>
</tr>
<tr>
<td>Operating system platforms</td>
<td>27</td>
</tr>
<tr>
<td>Output parameters</td>
<td></td>
</tr>
<tr>
<td>CPI-C</td>
<td>132</td>
</tr>
<tr>
<td>Overhead, buffer</td>
<td>294</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>Parallel conversations</td>
<td>125</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>CPI-C calls in COBOL</td>
<td>273</td>
</tr>
<tr>
<td>Partial format</td>
<td></td>
</tr>
<tr>
<td>Send/receive</td>
<td>99</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Data structure</td>
<td>299</td>
</tr>
<tr>
<td>Name of program</td>
<td></td>
</tr>
<tr>
<td>With Enable_UTM_UPIC</td>
<td>147</td>
</tr>
<tr>
<td>Network address</td>
<td>322</td>
</tr>
<tr>
<td>New conversation</td>
<td>77</td>
</tr>
<tr>
<td>New local address</td>
<td>76</td>
</tr>
<tr>
<td>New remote address</td>
<td>76</td>
</tr>
<tr>
<td>Node</td>
<td>33</td>
</tr>
</tbody>
</table>
partner application
  set host name 241
  set IP address 243
  set port 249
  set transport selector 251
  set transport selector format 253
partner configuration
  coordination 343
partner name
  in upicfile 325, 333
partner_LU_name 90, 165, 318
partner_LU_name_length 88, 165
PartnerLUName
  UpicB.ocx 47
PartnerPort
  UpicB.ocx 48
pass on send authorization 94
password
  incorrect 228
  invalid 137, 187, 198, 228, 233
  set 227
Password (UpicB.ocx) 44
PATH 352
path name 366
PCMX 31, 322, 354
PCMX diagnostics 370
PEND FI 95, 97
PEND KP 97
PEND RS 358
permission to send 92
  receive 192, 203
PGPOOL 188, 204, 212
ping 361
PORT 89
port number 102 320
portability
  CPI-C programs 262
Prepare_To_Receive 182
  call 182
  overview 94
PrepareToReceive (UpicB.ocx) 53
program
  sign off 145
program interface
  CPI-C 132
  XATMI 284
program name
  specify 147
program unit
  start 94
properties of secondary return code
  set 271
properties of UpicB.ocx 41
  incorrect input 42
    LocalName 43
    rules 42
property
  GLOBAL NAME 343
property handlers 76
Property Pages window (UpicB.ocx) 62
PROTOCOL 328
  query 154
  set in program 218
PROTOCOL characteristic
  query 154
  set 218
PTERM name with UPIC local 148
public diagnostic function 82
Q
query
  CHARACTER_CONVERSION 161
    client context 151
    cursor position offset 163
    encryption level 156
    secondary information 167
    state of conversation 159
Quickstart Kit 352
R
README files 20
receive 185
  blocking 96
  data 185
  data and format identifier 196
  formats 98
  message 95
receive (cont.)
  message segment 95
  multiple 109
  non-blocking 97
  overview 94, 95
  partial format 99
receive buffer 95
Receive state 86, 182
Receive timer 255
Receive type
  set 258
Receive_Mapped_Data 196
  call 196
  overview 94, 95, 99
receive_timer 255
receive_type 88
received_length
  Receive 186
  Receive_Mapped_Data 197
ReceiveMsg (UpicB.ocx) 50
ReceiveTimer (UpicB.ocx) 46
recovery service (XATMI) 277
Red Hat 14
remote connection 31
remote port number 321
remote-service-name 297
request 275
requester 275
RequestForData (UpicB.ocx) 52
request-response
  configure 298
requirements for encryption 121
reset
  Receive timer 255
  with Receive 97
Reset state 86
resource bottleneck
  error due to 136
resource files 106
resources 351
  CPI-C program (Unix system) 355
  local 362
  partner 362
  UPIC local (Unix system) 356
RESTART 113, 117
restart
  client context 119
  without 113
  XATMI 277
Restart (UpicB.ocx) 54
restart program
  for CPI-C 117
RET 239
return code
  CPI-C 132
  XATMI 291
return_code 165
return_control 87
revent 291
RSA key 120, 121
RSA-KEY 89
S
sample program
  OCXTestContainer (Windows) 375
  side information entry (Windows) 373
  UpicAutClt (Windows) 375
  UpicAutSvr (Windows) 375
  upmulti (Windows) 374
  uptac (Windows) 373
  utp (Windows) 373
SD 324
secondary information 167
  query 167
secondary return code
  query 170
  set properties 271
security attributes 76
security functions 114
security_password 88, 227
security_password_length 88
Security_Type 230
security_type 88
  error in UTM 137
security_user_ID 88
security_user_ID_length 88
semaphore 40
send
  data with format identifier 210
  format identifier 210
  formats 98
  message 93
  multiple 109
  partial format 99
  state 182
send authorization 86
  pass on 94
send buffer
  local 94
  transfer to server 94
send direction 192, 203
Send state 86
Send_Data 207
  overview 93
send_length
  Send_Mapped_Data 210
Send_Mapped_Data 210
  call 210
  overview 93, 99
send_type 87
SendMsg (UpicB.ocx) 49
sequence of CPI-C calls 91
server 25
service
  define 296
  XATMI 275
service context 117
service restart 113, 117
service state
  read 175
set
  deallocate_type 236
  encryption level 220
  host name of the partner application 241
  IP address of the partner application 243
  partner_LU_name 246
  password 227
  port for partner application 249
  Receive type 258
  synchronization level 261
  timeout timer 255
  TP_name 263
  TP_name_length 263
  transport selector for local application 267
  transport selector for partner application 251
  transport selector format for partner application 253
  transport selector format of the local application 269
  UTM function key 238
Set calls 132
Set_Allocate_Timer 213
Set_Client_Context 119
  call 215
Set_Communication_Protocol 218
Set_Conversation_Encryption_Level call 220
Set_Conversation_Security_Password 227
  call 227
  overview 114, 116
Set_Conversation_Security_Type 230
  overview 114
Set_Conversation_Security_User_ID 232
  overview 114
Set_Convetion 234
Set_Deallocate_Type 236
Set_Function_Key 238
  call 238
  overview 102
Set_Partner_Host_Name 321
Set_Partner_Host_Name call 241
Set_Partner_IP_Address 321
Set_Partner_IP_Address call 243
Set_Partner_LU-Name 246
Set_Partner_Port 321
Set_Partner_Port call 249
Set_Partner_Tsel 321
Set_Partner_Tsel call 251
Set_Partner_Tsel_Format 321
Set_Partner_Tsel_Format call 253
Set_Receive_Timer 255
  call 255
Set_Receive_Type 258
  call 258
  overview 96
Set_Sync_Level 261
   call 261
Set_TP_Name 263
SFUNC
   RET 239
shared memory
   UPIC local (Unix system) 356
shell variable
   UPICLOG 364
   UPICPATH 323
side information 86
side information entry
   for sample programs (Windows) 373
side information file 86, 323
SIGHUP 355
SIGINT 355
sign off
   CPI-C program 145
   XATMI 285
   XATMI client 288
sign on
   after password expiry 116
   CPI-C program 147
   multiple to UTM 149
   XATMI 285
signals 355
SIGNON service 113
SIGQUIT 355
single-step service
   communication with 108
size of UPIC trace file 366
socket interface 31
Solaris 14
Specify_Local_Port 321
Specify_Local_Port call 265
Specify_Local_Tsel 321
Specify_Local_Tsel call 267
Specify_Local_Tsel_Format 321
Specify_Local_Tsel_Format call 269
Specify_Secondary_Return_Code
   set 271
standalone UTM application 11
start
   CPI-C program 349, 354
   CPI-C program under BS2000 357
   program unit 94
   thread 125
   UTM service 93
Start state 86
starter-set calls 132
state
   query conversation state 159
   Receive, change to 182
   Send 192
   Send, change to Receive 182
state "Initialize" 86
state "Receive" 86
state "Reset" 86
state "Send" 86
state "Start" 86
state of a conversation 86
state table 386
status_received
   Receive_Mapped_Data 197
structure
   CPI-C application 91
subtypes 282
SUSE 14
SVCU 296
sym_dest_name
   Initialize_Conversation 179
Symbolic Destination Name 319
symbolic destination name 90, 319, 323, 324
   cluster 332
   length 132, 325, 333
   openUTM cluster 332
Symbolic_Destination_Name 70
SymDestName (UpicB.ocx) 46
synchronization level
   set 261
synchronous request-response paradigm 278
syntax
   LCF definition file 295
system access control functions 114
Index

T
T.61 character set 385
T.61 string 282
TAC
  encryption 123
TCP/IP 320
    check connection 361
TCP/IP port
      local application 265
      partner application 249
telnet 361
TEMP 363, 366
terminate conversation 141
    after transaction end 143
terminate thread 126
TestContainer, see OCXTestContainer
thread 125
  start 125
  terminate 126
thread-capable systems 70
threading 354
TIMEOUT 273
timeout 213
    Receive 188
timeout timer
    overview 96
    set for blocking Receive 255
timer
  UTM application 96
TMP 363, 366
TNS directory 322
TNS entry 322, 343
    for sample programs (Windows) 373
    Windows 373
TNS name 147
TP_name 88, 90
    set 263
TP_name_length 88
    set 263
tpacall 279
tpcall 278, 374
tpcall.ldf, local definition file 374
tpconnect 280
tpdiscon 280
TPEVENT 291
tperno 291
tpgetrply 279
tpinit 285, 286
  local name 305
TPName (UpicB.ocx) 44
TPOOL 131
tprecv 280
tpsend 280
tpterm 285, 288
trace
  UPIC 364
Trace (UpicB.ocx) 47
trace file
    directory 366
    name in XATMI 312
    name with UPIC 367
    size with UPIC 366
    size with XATMI 312
    UPIC 367
    XATMI 312
transaction code 90
CUpic 77
    in upicfile 326, 333
    invalid 137, 187
    XATMI 297
Transaction Server 37
transaction state 112
    read 175
TRANSDATA 320
transport connection
  error in 136
transport selector 320
    format for partner application 253
    format for the local application 269
    local application 267
    partner application 251
transport systems 320
T-SEL 89
T-SEL-FORMAT 89
two 318
two-level
    partner name 333
two-level partner name 325
two-part partner_LU_name 318, 320
  type
    security 230
typed buffer 275, 281, 292
    rules 292
    types 281
typed record 281
types
    XATMI 281

U
ULS 115
Unix platform 14
UPIC (Windows) sample programs 371
UPIC buffer 93
UPIC carrier system 27, 32
UPIC client 318
UPIC communication protocol
  specify in upicfile 328
UPIC conversations 26, 70
UPIC generation 323
  XATMI 303
UPIC local 30, 148
  Enable_UTM_UPIC 148
  linking in XATMI 307
  non-blocking Receive 97, 185
  partner name 325
  Set_Receive_Timer 255
  Set_Receive_Type 258
  Unix system 356
  UTMPATH 149
  Windows 351
UPIC log file 360, 363
UPIC remote 31
  linking in XATMI 308
  partner name 325
UPIC specific
  conversation characteristics 89
UPIC trace 364
  activate 365
  BS2000 365
  deactivate 369
  edit 369
UPIC.H 349
  BS2000 357
  Unixsystem 354
UPIC.INI 350, 363
UPIC.UpicCtrl.1 40
UpicAutClit, sample program for Windows 375
UpicAutSrv.exe 35
UpicAutSvr, sample program for Windows 375
UpicAutSvr.exe 68
UpicB.ocx 35
  advantages 37
  application example 64
  example in Visual Basic 38
  interfaces 40
  methods 48
  properties 41
  Property Pages window 62
UpicB.ocx methods
  ConvertIncoming 59
  ConvertOutgoing 60
  Disconnect 55
  GetLastError 58
  IncompleteDataReceived 57
  InConversation 55
  InSendingState 56
  PrepareToReceive 53
  ReceiveMsg 50
  RequestForData 52
  Restart 54
  SendMsg 49
UpicB.ocx properties
  BlockingReceive 45
  FunctionKey 43
  MapName 45
  MapOn 45
  Password 44
  ReceiveTimer 46
  SymDestName 46
  TPName 44
  Trace 47
  UserID 44
upic-cob 375
UPICFIL 323, 348
UPICFILE 347
Index

upicfile 323
  Enable_UTM_UPIC 147
  local name 305
XATMI 303
upicfile entry
  UTM cluster application 332
UPIC-L 32, 319
UPICL 318
UPIC-L configuration 319
UPILOG 347, 348, 363, 364
UPICPATH 323, 347
UPIC-R 320
UPICR 318
UPIC-R (Windows) 32
UPIC-R configuration 320
UPIC-R with CMX 322
UpicSimpleClient 376
UPIC-specific functions 135
UPICTRA 348
UPICTRACE 347, 364
upicw32.lib 349
upicws32.lib 349
upmulti, sample program for Windows 374
uptac, sample program for Windows 373
usage of typed buffer 292
use of TNS entries 343
USER 304
user buffer 281
user concept of openUTM 113
user data
  encrypt 120
user ID 187
  invalid 187, 198
  multiple sign-ons 115
UserID (UpicB.ocx) 44
UTM 319
UTM cluster application 11, 33
  cluster administration journal 399
UTM directory, UPIC local (Unix system) 356
UTM error message 363
UTM formats
  receive 196
  send 210
UTM function key 102
  set 238
UTM server 318
  communication with 107
UTM service
  addressing 90
  service start 93
UTM user concept 113
UTM user ID
  invalid 187, 198
  not generated 137
  set 232
UTM version
  below V4.0 200, 211, 239
  earlier than V4.0, UPIC protocol for 328
  V4.0 and higher, UPIC protocol for 328
UTM.local 319
utmcnv32.def 106
utmcnv32.rc, resource.h 106
UTMPATH 180
  not set 149
utp32, Windows 32-bit sample program 373
V
Visual Basic 35
  use of UpicB.ocx 38
Visual C++, configuring with 352
void GetMapName 77
void GetTPName 77
void Reset 82
void SetEncryption 76
void SetFunctionKey 77
void SetHost 74
void SetLocal 76
void SetMapName 77
void SetPort 72, 74
void SetRemote 76
void SetSecurity 76
void SetTPName 77
void SetTselName 72, 74
W
wait time
  maximum 203
Windows registry 38
Windows system 14
Windows, UPIC local 351
WINDOWS.H 349
without restart 113
worker thread 70
wrapper class 69

X
X/Open
  CPI-C interface 381
X_C_TYPE 281, 282
  conversion 283
X_COMMON 281, 282
  conversion 283
X_OCTET 281
  BUFFERS operand 298
XATMI 275
  last output message 117
  program interface 284
XATMI application 28
XATMI client, sign off 288
XATMI program 28
  access to services 29
XATMI U-ASE 276, 281
XATMIGEN 300
XTSVRTR 309