English

Fujitsu Software BS2000

openUTM-Client for the UPIC Carrier System

Client-Server Communication with openUTM

User Guide

Valid for:
openUTM-Client V7.0A25

Edition November 2022

Comments… Suggestions… Corrections…

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

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com.

Certified documentation according to DIN EN ISO 9001:2015

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 9001:2015.

Copyright and Trademarks

Copyright © 2025 Fujitsu

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

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

Table of Contents

Client-Server Communication with openUTM . . . 8
1 Preface . . . 9
1.1 Brief description of the openUTM-Client product . . . 11
1.2 Summary of contents and target group . . . 12
1.3 Summary of contents of the openUTM documentation . . . 13
1.3.1 openUTM documentation . . . 14
1.3.2 Documentation for the openSEAS product environment . . . 17
1.3.3 Readme files . . . 18
1.4 Changes since the last version of this manual . . . 19
1.5 Notational conventions . . . 20
2 Application area . . . 22
2.1 The concept of openUTM-Client . . . 23
2.2 Client/server communication with openUTM . . . 25
2.3 UPIC local, UPIC remote and multithreading . . . 26
2.3.1 UPIC remote . . . 27
2.3.1.1 Distribution of communication over multiple communication end points . . 28
2.3.1.2 Setting up a list of communication end points . . . 29
2.3.2 UPIC local (Unix, Linux and Windows systems) . . . 30
2.3.3 Multithreading . . . 31
2.4 Support for UTM cluster applications on Linux-, Unix- and Windows-Systems 32
3 CPI-C interface . . . 33
3.1 CPI-C terms . . . 34
3.2 General structure of a CPI-C application . . . 38
3.3 Exchange of messages with a UTM service . . . 39
3.3.1 Sending a message and starting a UTM service . . . 40
3.3.2 Receiving a message, blocking and non-blocking receive . . . 42
3.3.3 Sending and receiving formats . . . 44
3.3.4 UTM function keys . . . 47
3.3.5 Cursor position . . . 48
3.3.6 Code conversion . . . 49
3.3.6.1 Standard code conversion tables . . . 50
3.3.6.2 Modifying code conversion tables on Unix and Linux systems . . . 52
3.3.6.3 Modifying code conversion tables on Windows systems . . . 53
3.3.6.4 Modifying code conversion tables on BS2000 systems . . . 54
3.4 Communicating with an UTM application . . . 55
3.4.1 Communicating in a single-step UTM service . . . 56

3.4.2 Communicating in a multi-step UTM service 58
3.4.3 Communicating in a multi-step UTM service with distributed transaction processing 59
3.4.4 Querying the transaction state 60
3.5 User concept, security and restart 61
3.5.1 User concept 62
3.5.2 Security functions 63
3.5.3 Restart 65
3.6 Encryption 68
3.7 Multiple conversations (Unix, Linux and Windows systems) 72
3.8 DEFAULT server and DEFAULT name of a client 76
3.8.1 Multiple connections to the same UTM application with the same name 77
3.9 CPI-C calls in UPIC 78
3.9.1 Overview 79
3.9.2 Allocate - Establishing a conversation 82
3.9.3 Convert_Incoming - Converting data from code of sender to local code 85
3.9.4 Convert_Outgoing - Converting data from local code to code of receiver . . . 86
3.9.5 Deallocate - Terminating a conversation 87
3.9.6 Deferred_Deallocate - Terminating a conversation after termination of a transaction 89
3.9.7 Disable_UTM_UPIC - Signing off from the UPIC carrier system 91
3.9.8 Enable_UTM_UPIC - Signing on to the UPIC carrier system 93
3.9.9 Extract_Client_Context - Querying the client context 96
3.9.10 Extract_Conversation_Encryption_Level - Querying encryption level 99
3.9.11 Extract_Conversation_State - Querying state of conversation 102
3.9.12 Extract_Convertion - Querying the value of the CHARACTER_CONVERTION conversation characteristic 104
3.9.13 Extract_Cursor_Offset - Querying cursor position offset 106
3.9.14 Extract_Max_Partner_Index - Querying the maximum index of partner applications 108
3.9.15 Extract_Partner_LU_Name - Querying partner_LU_Name 110
3.9.16 Extract_Partner_LU_Name_Ex - Querying full length partner_LU_Name . . 112
3.9.17 Extract_Secondary_Information - Querying secondary information 114
3.9.18 Extract_Secondary_Return_Code - Querying secondary return codes 116
3.9.19 Extract_Shutdown_State - Querying the shutdown state of the server 120
3.9.20 Extract_Shutdown_Time - Query the shutdown time of the server 122
3.9.21 Extract_Transaction_State - Querying service and transaction state of the server 125
3.9.22 Initialize_Conversation - Initializing the conversation characteristics 128
3.9.23 Prepare_To_Receive - Changing state from “Send” to “Receive” 131
3.9.24 Receive - Receiving data from a UTM service 133

3.9.25 Receive_Mapped_Data - Receiving data and format identifier from a UTM service 142
3.9.26 Send_Data - Sending data to a UTM service 151
3.9.27 Send_Mapped_Data - Sending data and format identifier 153
3.9.28 Set_Allocate_Timer - Setting timer for the allocate call 156
3.9.29 Set_Client_Context - Setting the client context 158
3.9.30 Set_Conversation_Encryption_Level - Setting the encryption level 160
3.9.31 Set_Conversation_Security_New_Password - Setting new password 163
3.9.32 Set_Conversation_Security_Password - Setting the password 165
3.9.33 Set_Conversation_Security_Type - Setting the security type 167
3.9.34 Set_Conversation_Security_User_ID - Setting the UTM user ID 169
3.9.35 Set_Convertion - Setting the CHARACTER_CONVERTION conversation characteristic 171
3.9.36 Set_Deallocate_Type - Setting deallocate_type 173
3.9.37 Set_Function_Key - Setting a UTM function key 175
3.9.38 Set_Partner_Host_Name - Setting the partner host name 177
3.9.39 Set_Partner_Index - Setting the partner application index 179
3.9.40 Set_Partner_IP_Address - Setting the IP address of the partner application . . . 181
3.9.41 Set_Partner_LU_Name - Setting the conversation characteristics partner_LU_name 184
3.9.42 Set_Partner_Port - Setting the TCP/IP port for the partner application 186
3.9.43 Set_Partner_Tsel - Setting the T-SEL of the partner application 188
3.9.44 Set_Partner_Tsel_Format - Setting the T-SEL format of the partner application 190
3.9.45 Set_Receive_Timer - Setting the timer for a blocking receive 192
3.9.46 Set_Receive_Type - Setting the receive type 194
3.9.47 Set_Sync_Level - Setting a synchronization level 196
3.9.48 Set_TP_Name - Setting TP-name 198
3.9.49 Specify_Local_Port - Setting the TCP/IP port of the local application 200
3.9.50 Specify_Local_Tsel - Setting the T-SEL of the local application 202
3.9.51 Specify_Local_Tsel_Format - Setting the TSEL format of the local application . 204
3.9.52 Specify_Secondary_Return_Code - Setting the properties of the secondary return code 206
3.10 COBOL interface 208
4 XATMI interface 210
4.1 Linking client/server applications 211
4.1.1 Default server 212
4.1.2 Restart 213
4.2 Communication paradigms 214
4.3 Typed buffers 217

4.4 Program interface . . . **220**
4.4.1 XATMI functions for clients . . . 221
4.4.2 Calls for connecting to the carrier system . . . 223
4.4.2.1 tpinit - Initializing the client . . . 224
4.4.2.2 tpterm - Signing the client off . . . 226
4.4.3 Transaction control . . . 227
4.4.4 Mixed operation . . . 228
4.4.5 Administration interface . . . 229
4.4.6 Header files and COPY elements . . . 230
4.4.7 Events and error handling . . . 231
4.4.8 Creating typed buffers . . . 232
4.4.9 Characteristics of XATMI in UPIC . . . 234
4.5 Configuring . . . **235**
4.5.1 Creating the local configuration file . . . 236
4.5.2 The xatmigen tool . . . 240
4.5.3 Configuring the carrier system and UTM partners . . . 243
4.5.3.1 Configuring UPIC . . . 244
4.5.3.2 Initialization parameters and UTM configuration . . . 245
4.6 Running XATMI applications . . . **248**
4.6.1 Linking and starting an XATMI program . . . 249
4.6.1.1 Linking an XATMI program on Windows systems . . . 250
4.6.1.2 Linking an XATMI program on Unix and Linux systems . . . 251
4.6.1.3 Linking an XATMI program on BS2000 systems . . . 252
4.6.1.4 Starting the program . . . 253
4.6.2 Setting Environment variables on Unix, Linux and Windows systems . . . 254
4.6.3 Setting job variables on BS2000 systems . . . 256
4.6.4 Trace . . . 258
4.7 xatmigen messages . . . **259**
5 Configuration . . . **262**
5.1 Configuration without upicfile . . . **263**
5.1.1 UPIC-R configuration . . . 265
5.1.2 UPIC-L configuration (Unix, Linux and Windows systems) . . . 267
5.1.3 Configuration using BCMAP entries (BS2000 systems) . . . 268
5.2 The side information file (upicfile) . . . **269**
5.2.1 Side information for standalone UTM applications . . . 270
5.2.2 Side information for list of partner applications . . . 275
5.2.3 Side information for UTM cluster applications . . . 276
5.2.4 Side information for the local application . . . 281
5.3 Coordination with the partner configuration . . . **284**
6 Implementing CPI-C applications . . . **287**
6.1 Runtime environment, linking, starting . . . **288**

6.1.1 Implementing on Windows systems 290
6.1.1.1 Compilation, linking, starting on Windows systems 291
6.1.1.2 Runtime environment, environment variables on Windows systems 292
6.1.1.3 Special features of implementing UPIC local on Windows systems 293
6.1.2 Implementation on Unix and Linux systems 295
6.1.2.1 Compilation, linking, starting on Unix and Linux systems 296
6.1.2.2 Runtime environment, environment variables on Unix and Linux systems . . . 297
6.1.2.3 Special features when using UPIC local on Unix and Linux systems 298
6.1.3 Using on BS2000 systems 299
6.2 Handling of CPI-C partners by openUTM 300
6.3 Behavior in the event of errors 301
6.4 Diagnostics 304
6.4.1 UPIC log file 305
6.4.2 UPIC trace 306
6.4.3 PCMX diagnostics (Windows systems) 310
7 Examples 311
7.1 Sample programs for Windows systems 312
7.1.1 uptac (Windows systems) 313
7.1.2 utp32 (Windows systems) 314
7.1.3 tpcall (Windows systems) 315
7.1.4 upic-cob (Windows systems) 316
7.2 UpicAnalyzer and UpicReplay on 64-bit Linux systems 317
7.2.1 UpicAnalyzer (64-bit Linux systems) 318
7.2.2 UpicReplay (64-bit Linux systems) 319
7.3 Configuration UPIC on Windows systems <-> openUTM on BS2000 systems . 321
7.3.1 Configuration on the Windows system 322
7.3.2 UTM Configuration on the BS2000 system 323
7.4 Configuration UPIC on Windows systems <-> openUTM on Unix or Linux systems 324
7.4.1 UPIC Configuration on the Windows system 325
7.4.2 UTM Configuration on the Unix or Linux system 326
8 Appendix 327
8.1 Differences between the X/Open CPI-C interface 328
8.2 Character sets 330
8.3 State table 332
9 Glossary 339
10 Abbreviations 372
11 Related publications 377

Client-Server Communication with openUTM

1 Preface

The IT infrastructure of today's companies as the heart and engine of the business must meet the requirements of the digital age. At the same time, it has to cope with increased amounts of data as well as with stricter requirements from the environment, e.g. compliance requirements. It must also be possible to integrate additional applications at short notice. And all this under the aspect of guaranteed security.
Thus, essential requirements for a modern IT infrastructure consist of, among others

- Flexibility and almost limitless scalability also for future requirements
- high robustness with highest availability
- absolute safety in all respects
- Adaptability to individual needs
- Causing low costs

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

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

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

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

- consistency of data and processing
- high availability of the applications
- 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

A UTM application on Unix, Linux and Windows systems can be run as a standalone UTM application or sumultanously on several different computers as a UTM cluster application.

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

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

The products BeanConnect and WebTransactions are briefly presented in the performance overview. There are separate manuals for these products.

> **i** Wherever the term Linux system or Linux platform is used in the following, then this should be understood to mean a Linux distribution such as SUSE or Red Hat.
>
> Wherever the term Windows system or Windows platform is in the following, this should be understood to mean all the variants of Windows under which openUTM runs.
>
> Wherever the term Unix system or Unix platform is used in the following, then this should be understood to mean a Unix-based operating system such as Solaris or HP-UX.

1.1 Brief description of the openUTM-Client product

The product openUTM-Client offers client/server communication with openUTM server applications which run on Unix, Linux and Windows systems and on BS2000 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, XATMI and TX. CPI-C, XATMI and TX are defined in the corresponding X/Open specifications, see chapter „Related publications“ starting on "Related publications".

TX is supported by the OpenCPIC carrier system. CPI-C and XATMI are supported by both the UPIC and the OpenCPIC carrier systems:

- CPI-C stands for **C**ommon **P**rogramming **I**nterface for **C**ommunication.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.

openUTM-Client for different platforms

openUTM-Client is available for the following platforms:

- Windows systems
- Unix and Linux systems
- BS2000 systems (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.

> **i** Wherever the term Unix system is used in the following, then this should be understood to mean a Unix-based operating system such as Solaris or HP-UX.
> Wherever the term Linux system is used in the following, then this should be understood to mean a Linux distribution such as SUSE or Red Hat.
>
> Wherever the term Windows system or Windows platform is used below, this should be understood to mean all the variants of Windows under which openUTM runs.

1.2 Summary of contents 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 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 the Windows platforms, Unix platforms, Linux platforms and BS2000 systems.

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, the online help for the graphical administration workstation openUTM WinAdmin and the graphical administration tool WebAdmin as well as release notes.
There are manuals and release notes that are valid for all platforms, as well as manuals and release notes that are valid for BS2000 systems and for Unix, Linux and Windows systems.

All the manuals and release notes are available on the internet at https://bs2manuals.ts.fujitsu.com.

The following sections provide a task-oriented overview of the openUTM V7.0 documentation.

You will find a complete list of documentation for openUTM in the chapter on related publications at the back of the manual.

Introduction and overview

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

Programming

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

Configuration

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

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

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

Linking, starting and using UTM applications

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

Administering applications and changing configurations dynamically

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

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

Testing and diagnosing errors

You will also require the **Messages, Debugging and Diagnostics** manuals (there are separate manuals for Unix, Linux and Windows systems and for BS2000 systems) to carry out the tasks mentioned above. These manuals describe how to debug a UTM application, the contents and evaluation of a UTM dump, the 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. 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 **openUTM-Client for the OpenCPIC Carrier System** manual describes how to install and configure OpenCPIC and configure an OpenCPIC application. It indicates what needs to be taken into account when programming a CPI-C application and what restrictions apply compared with the X/Open CPI-C interface.
- The documentation for the product **openUTM-JConnect** shipped with **BeanConnect** consists of the manual and a 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**.
- You can also use the software product WS4UTM (WebServices for openUTM) to provide services of UTM applications as Web services. To do this, you need the **Web Services for openUTM** manual. Working with the graphical user interface is described in the corresponding **online help system**.

Communicating with the IBM world

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

PCMX documentation

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

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

1.3.2 Documentation for the openSEAS product environment

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

Integrating Java EE application servers and UTM applications

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

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

Connecting to the web and application integration

Alternatively, you can use the WebTransactions product instead of the UTM HTTP program interface. Then you will need the **WebTransactions** manuals. The manuals will also be supplemented by JavaDocs.

1.3.3 Readme files

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

Readme files are available to you online in addition to the product manuals under the various products at https://bs2manuals.ts.fujitsu.com.

Information on BS2000 systems

The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product's files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at https://bs2manuals.ts.fujitsu.com.

1.4 Changes since the last version of this manual

The manual openUTM-Client V7.0 for the UPIC Carrier System covers the following no functional changes since the manual openUTM-Client V6.5 for the UPIC Carrier System:

Encryption

The encryption functionality in openUTM-Client has been revised. Security gaps have been closed, modern methods have been adopted and delivery has been simplified as follows:

- UTM-CLIENT-CRYPT variant
  Until now, the encryption functionality in openUTM-Client was only available if the product UTM-CLIENT-CRYPT was installed. With openUTM Client V7.0 this is no longer necessary. As of this version, it is decided at runtime whether the encryption functionality is available or not.
- Security
  A vulnerability has been fixed when communicating with a UTM application.
- Encryption Level 5
  The o penUTM client V7.0 supports communication with UTM V7.0 applications when ENCRYPTION-LEVEL 5 was generated for the connections to the UPIC client.
  With Level 5 the Diffie-Hellman method, based on Elliptic Curves, is used to agree on the session key. Input /output messages are encrypted using the AES-GCM algorithm. AES-GCM is an authenticated encryption algorithm designed to provide both data authenticity (integrity) and confidentiality.
  Level 5 is supported by the openUTM-Client on all platforms.
- Encryption BS2000
  openUTM-Client (BS2000) uses openSSL instead of BS2000-CRYPT analogous to Unix, Linux and Windows systems.

1.5 Notational conventions

Metasyntax

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

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

`...`	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`

Symbols

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

Indicates notes that are of particular importance.

Indicates warnings.

Other

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

filebase On Unix, Linux and Windows systems, designates the directory of the UTM application. This is the base name generated in the KDCDEF statement MAX KDCFILE=.

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

upic_dir The directory under which UPIC Client for UPIC Carrier System is installed on Unix, Linux, or Windows system.

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 for load simulations of UTM applications.

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.

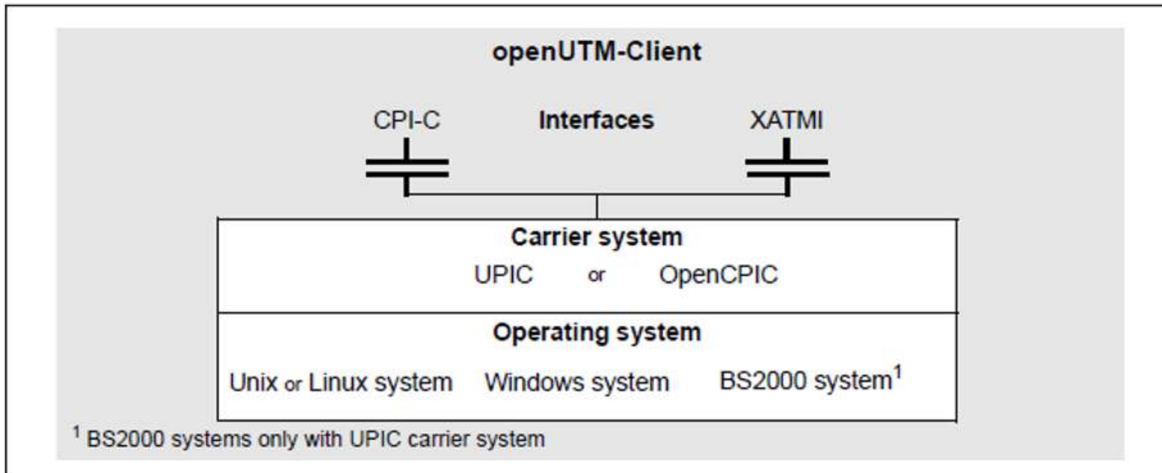


Figure 1: Standardized X/Open interfaces

Interfaces

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

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 Unix, Linux and Windows systems or BS2000 systems, PCMX in Unix, Linux and Windows systems or BCAM in BS2000 systems).

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
- Unix and Linux systems
- BS2000 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 an UTM server application.

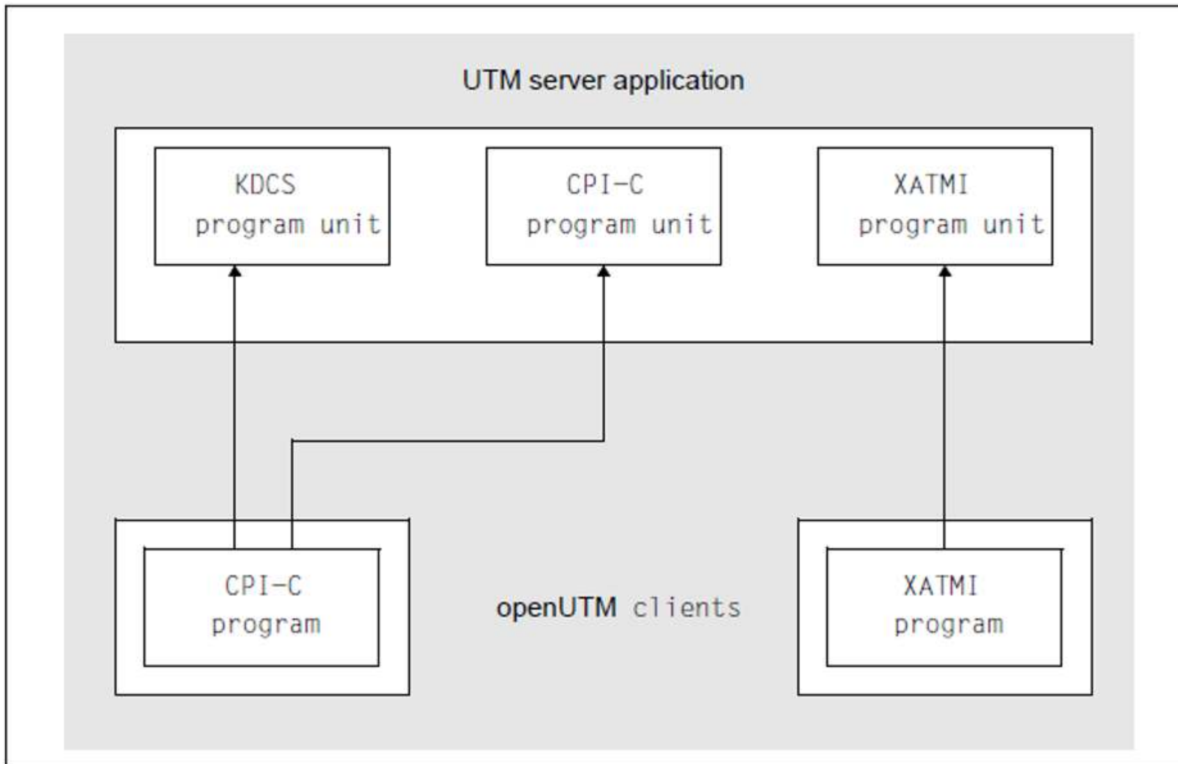


Figure 2: Interfaces between openUTM server and openUTM Clients

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.

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, Linux and Windows systems) and UPIC remote (all platforms)

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

2.3.1 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 (Unix, Linux and Windows systems and BS2000 systems). You need the product openUTM-Client for this. openUTM-Client contains two different versions of UPIC remote. In one variant, TCP/IP is used 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 3 (UPIC remote)).

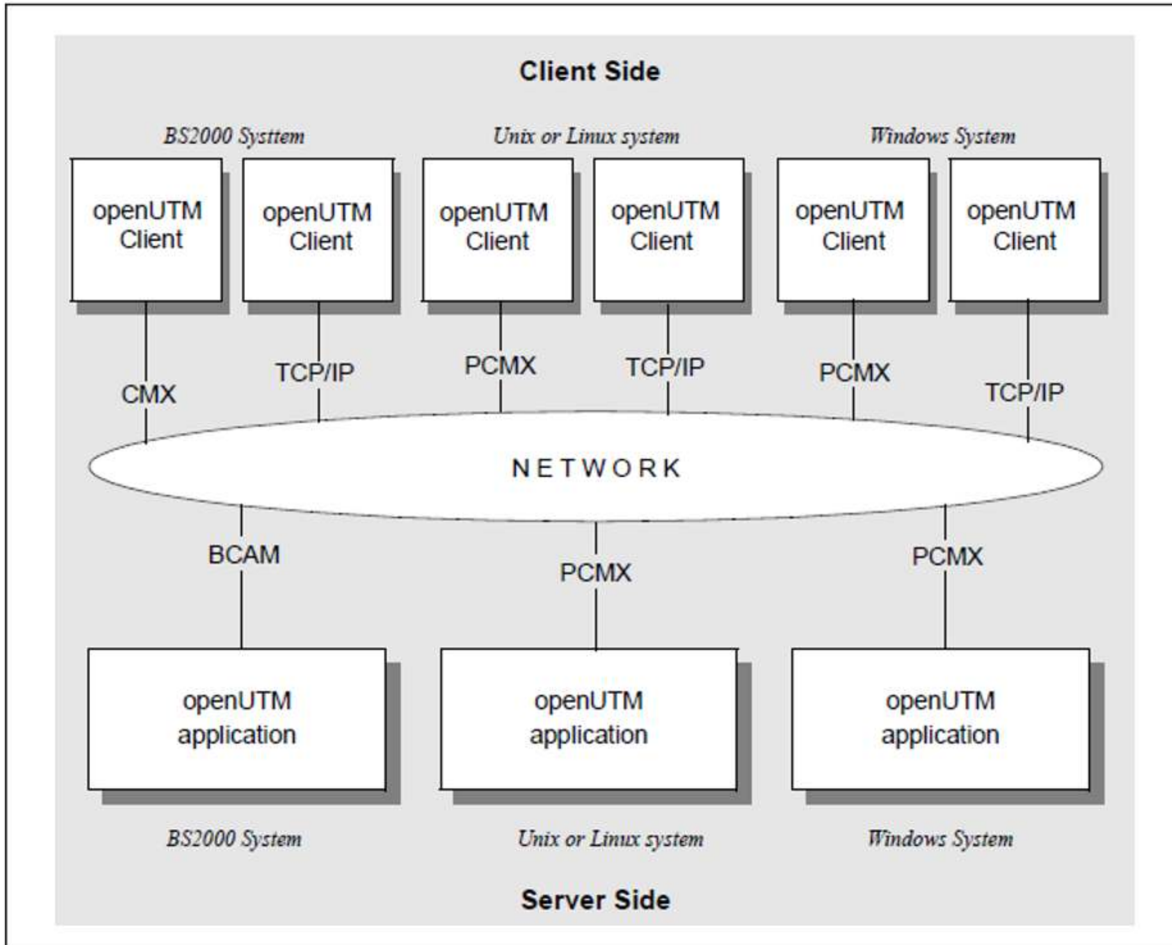


Figure 3: Remote connection to UTM applications

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 the openUTM application is handled by the communication components TCP/IP or PCMX.

2.3.1.1 Distribution of communication over multiple communication end points

UPIC-Remote enables communication and thus load to be distributed over multiple communication end points. This allows the implementation of “UPIC routing”. For example, if a very large number of clients (more than 1000) are communicating with a standalone UTM application on a Unix, Linux, or Windows system, it may be necessary to distribute the clients over multiple communication end points (BCAMAPPLs) in the UTM application. It is even possible to distribute communication over multiple standalone UTM applications. However, due to code conversion these should all be running on the same platform.

The client program requires a list of the associated communication end points for the UTM application(s). From this list, a random communication end point is then selected and is used to start the next communication. This random selection ensures client-side load balancing.

If communication is not possible with this selected communication end point, an attempt is automatically made to establish a connection with a different communication end point. Once again, this communication end point is selected randomly from the remaining entries in the list.

This process is repeated until a connection can be established with a communication end point for the UTM application or until it is detected that none of the communication end points from the list can be accessed.

The figure below shows a distribution of communication over three communication end points:

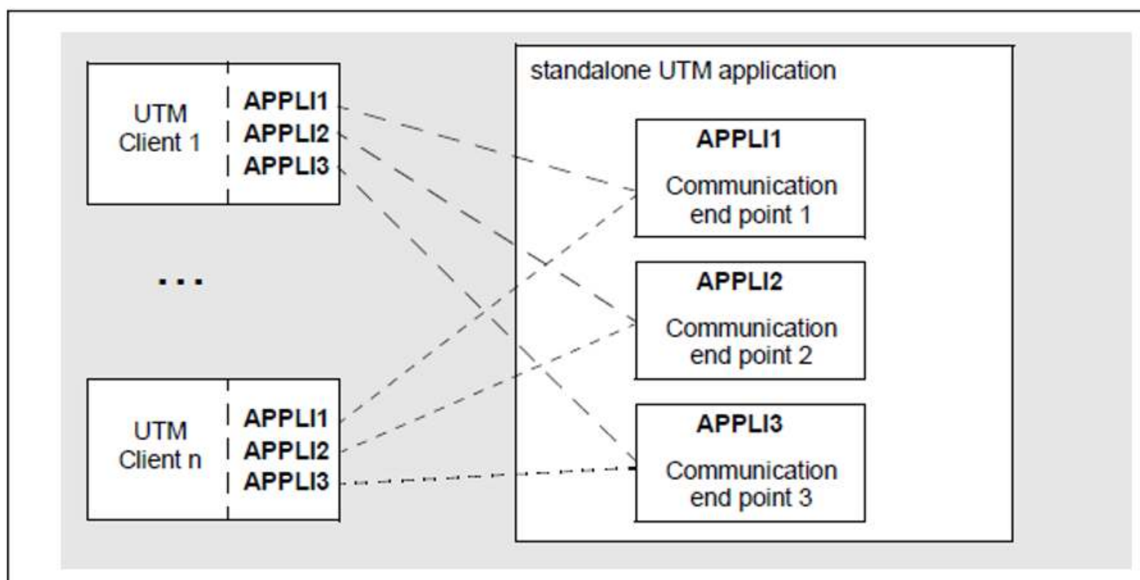


Figure 4: Communication of clients with multiple communication end points

2.3.1.2 Setting up a list of communication end points

The list of communication end points can be specified directly in the client program or passed to the side information file (`upicfile`).

How to transfer a list of communication end points using the `upicfile` is described in detail in section “Side information for list of partner applications”.

Setting up a list in the client program

To specify the list in the client program:

1. Select the first communication end point in the list using the call *Set_Partner_Index (CMSPIN)* with index 1. For details, see section “Set_Partner_Index - Setting the partner application index”.
2. Use subsequent *Set_Partner_xxx* calls to assign the appropriate addressing information to this communication end point.
3. For the next communication end point, repeat steps 1 and 2 with the index 2.
4. Repeat steps 1 and 2, incrementing the index, until the list is complete.

Note

- As long as no *Allocate* call has been executed yet, you can change the values of individual communication end points at any time.
- Once entries have been created, they cannot be deleted within a conversation.
- When a conversation is terminated, the list is automatically deleted and can be set up again after an *Initialize_Conversation* call.

2.3.2 UPIC local (Unix, Linux and Windows systems)

With UPIC local (UPIC-L), you can link a client program locally with a UTM application on the same Unix, Linux or Windows system. The UPIC-local carrier system is available for Unix, Linux 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 component PCMX.

This option is only available on a Unix, Linux or Windows system.

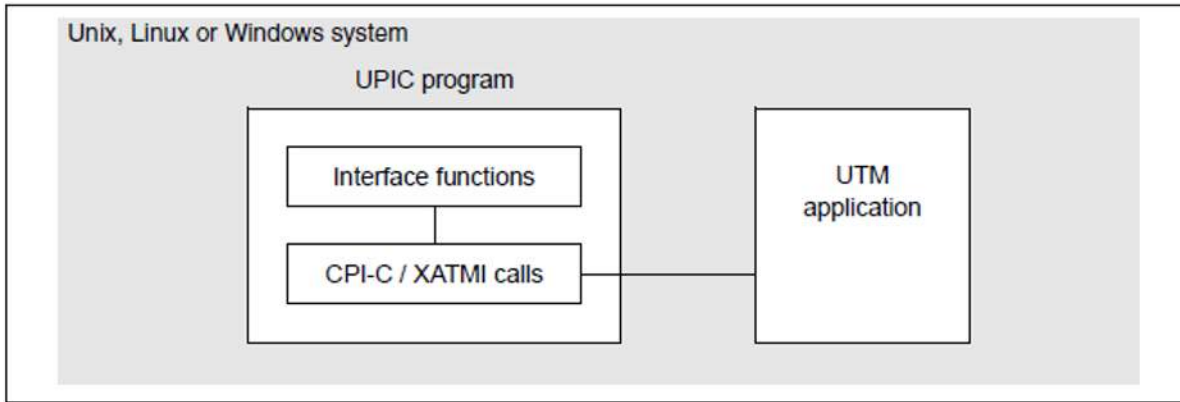


Figure 5: Local connection to a UTM application

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.

2.3.3 Multithreading

The UPIC carrier system is basically multithreading-capable. Whether you can use this capability in your application depends on the communication mode (local/remote) and the platform:

- UPIC-L on Unix, Linux and Windows systems is not multithreading-capable
- UPIC-R on Windows systems is multithreading-capable
- UPIC-R on Unix or Linux systems is multithreading-capable depending on the UPIC library which is used (`libupiccmx` , `libupicsoc` or `libupicsocmt`)
- UPIC-R on BS2000 systems is not multithreading-capable

2.4 Support for UTM cluster applications on Linux-, Unix- and Windows-Systems

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.

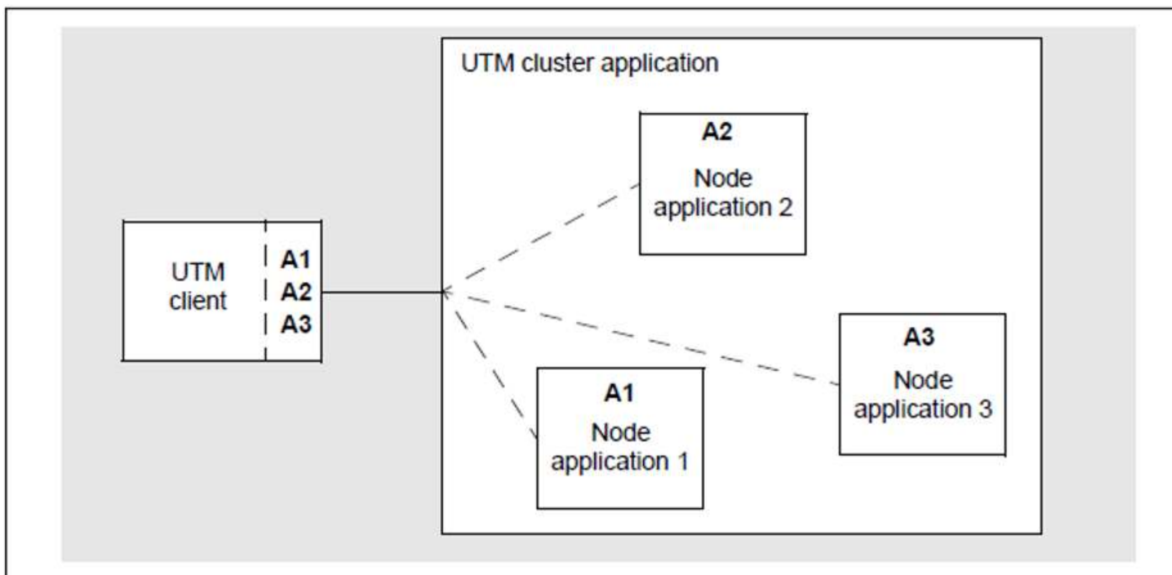


Figure 6: Communication with a UTM cluster application

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”.

3 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 Unix, Linux or Windows systems or BS2000 systems. 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.

3.1 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 Conversation Characteristics.
- 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:

State	Description
Start	The program is not signed on to the UPIC carrier system. (before the *Enable_UTM_UPIC* call or after the *Disable_UTM_UPIC* call).
Reset	No conversation is assigned to the conversation_ID.
Initialize	The *Initialize_Conversation* call was completed successfully and a *conversation_ID* was assigned to the conversation.
Send	The program is authorized to send data in the conversation.
Receive	The program can receive information via the conversation.

Table 1: 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”. A table of states can be found in the appendix on page State table. 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:

Conversation characteristics	Initialization value for Initialize_Conversation
conversation_type	CM_MAPPED_CONVERSATION
return_control	CM_WHEN_SESSION_ALLOCATED
send_type	CM_BUFFER_DATA
sync_level	CM_NONE

Table 2: Preset conversation characteristics

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

Conversation characteristics	Initialization value for Initialize_Conversation
deallocate_type	CM_DEALLOCATE_SYNC_LEVEL
partner_LU_name	Value from side information, dependent on the symbolic destination name
partner_LU_name_length	Length of *partner_LU_name*
receive_type	CM_RECEIVE_AND_WAIT
security_new_password	Empty
security_new_password_length	0
security_password	Blank
security_password_length	0
security_type	CM_SECURITY_NONE
security_user_ID	Blank
security_user_ID_length	0
TP_name	Value from side information, dependent on the symbolic destination name

TP_name_length	Length of *TP_name*

Table 3: 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:

Conversation characteristics	**Initialization value for Initialize_Conversation**
CHARACTER_CONVERTION	CM_NO_CHARACTER_CONVERTION
CLIENT_CONTEXT	empty
ENCRYPTION-LEVEL	0
PORT	102
T-SEL	Value derived from *partner_LU_name*
T-SEL-FORMAT	Value derived from *partner_LU_name*
HOSTNAME	Value derived from *partner_LU_name*
IP-ADDRESS	Not initialized
RSA-KEY	Allocated by the UTM application
SECONDARY_RETURN_CODE	CM_RETURN_TYPE_SECONDARY
TRANSACTION_STATE	empty

Table 4: UPIC specific conversation characteristics for remote applications

Values for local applications	**Initialization value for Enable_UTM_UPIC**
PORT	102
T-SEL	Value derived from local application name
T-SEL-FORMAT	Value derived from local application name

Table 5: 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_Conversation_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 UTM configuration. 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 UTM configuration is found in section “Coordination with the partner configuration”.
When a `upicfile` is used, this offers the advantage that ther UTM configuration can be modified (e.g. by moving the UTM server application to another system) without the client programs having to be modified.

3.2 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":

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* provides the runtime environment for the client.
2. After calling *Enable_UTM_UPIC*, you can use the *Specify_...* 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 "CPI-C terms".
4. After initialization you can set or modify various conversation characteristics using the *Set_...* calls (see the modifiable characteristics on "CPI-C terms").
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".

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 (Unix, Linux and Windows systems)".

3.3 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 application” 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).

3.3.1 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.

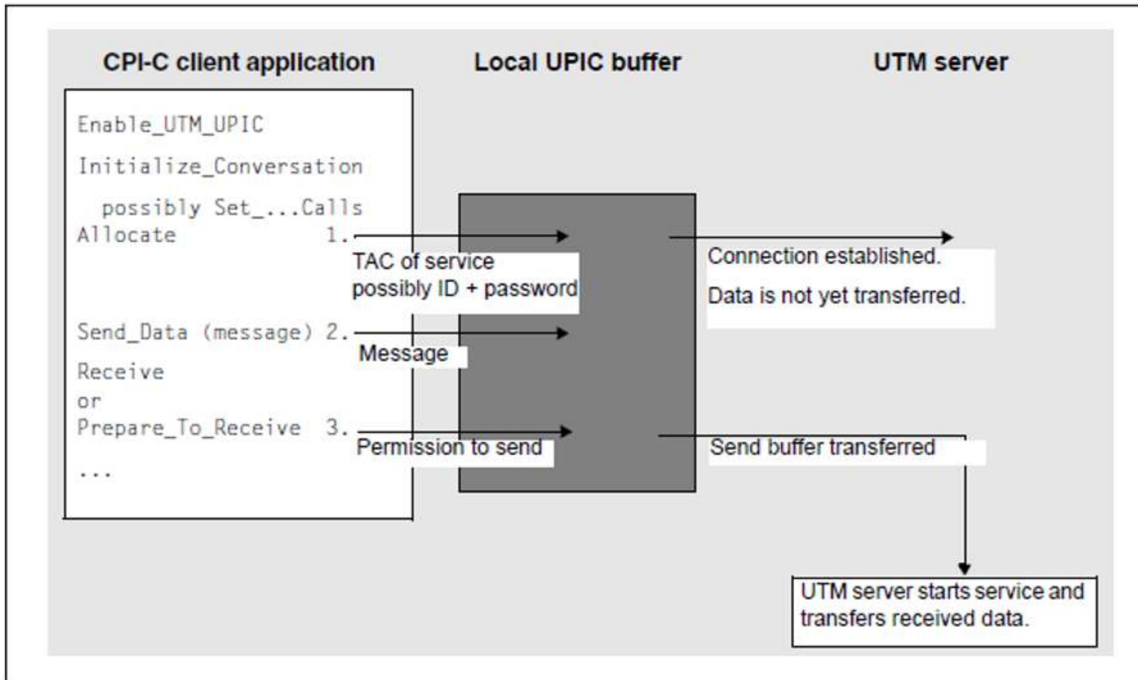


Figure 7: Client starts service in a UTM partner application

Explanation of the diagram

1. Following the *Allocate* call, the conversation is “established” and a connection to the UTM application 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”.

   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").

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.

3.3.2 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 application 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 (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.

> i 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.

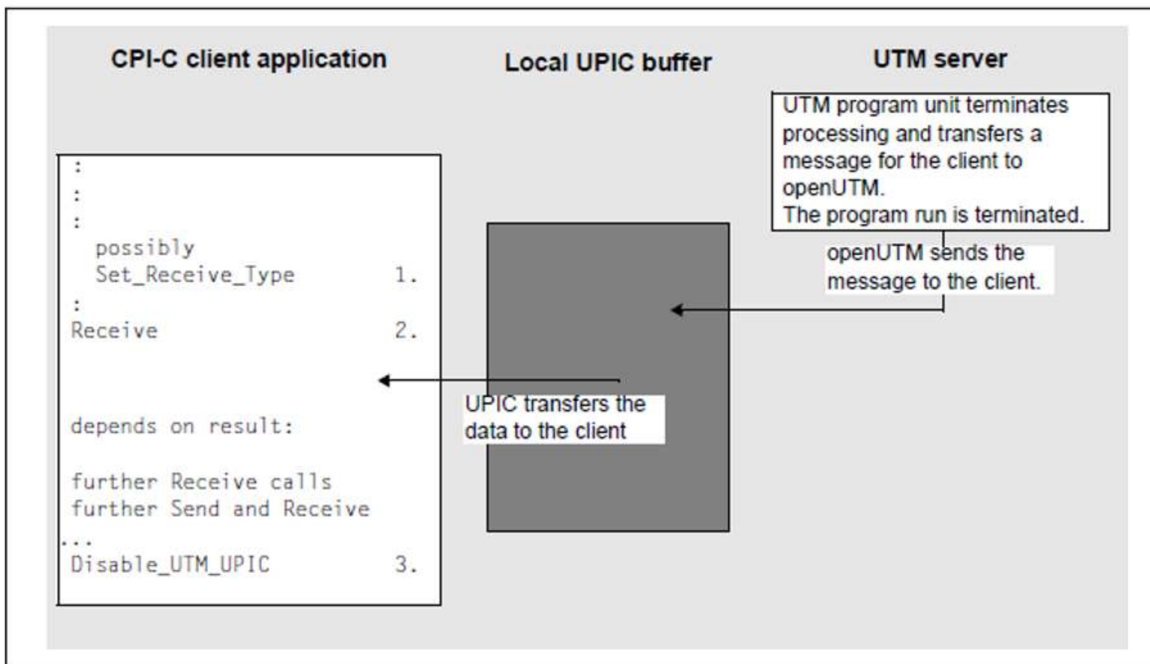


Figure 8: Client receives a message from server, conversation is shut down

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 situations can occur:
   - The *Receive*-call is issued and the UTM service has not terminated the conversation (*return_code*=CM_OK) and
     - the message segment could not be read completely (*data_received* =CM_INCOMPLETE_DATA_RECEIVED), because it is longer than the buffer provided.
       --> More parts of this message part must be read with a *Receive*- e.g. *Receive_Mapped_Data*-call.
     - the message segment could be read completely (*data_received*=CM_COMPLETE_DATA_RECEIVED), the client has not received the permission to send (*status_received*=CM_NO_STATUS_RECEIVED).
       --> More message parts must be read with a *Receive*- e.g. *Receive_Mapped_Data*-call.
     - the message segment could be read completely (*data_received*=CM_COMPLETE_DATA_RECEIVED), it was the last (or only) message part of the UTM program unit run, and the client has received the permission to send (*status_received*=CM_SEND_RECEIVED).
       --> The conversation must be continued by the client with at least one *Send*- e.g. *Send_Mapped_Data*-call and repeated *Receive*- e.g. *Receive_Mapped_Data*-calls. In this case this is a multi-step service (the service has terminated with PEND RE, PEND KP, PGWT KP or PGWT CM).
   - The *Receive*-call is issued and the UTM service has terminated the conversation (*return_code* =CM_DEALLOCATE_NORMAL - the service has terminated with PEND FI).
     --> The program switches to the "Reset" state. It can now establish a new conversation or sign off from UPIC with *Disable_UTM_UPIC()*.
3. Once the last conversation has terminated, the client program calls *Disable_UTM_UPIC* in order to sign off from UPIC.

3.3.3 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”.

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”.

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.

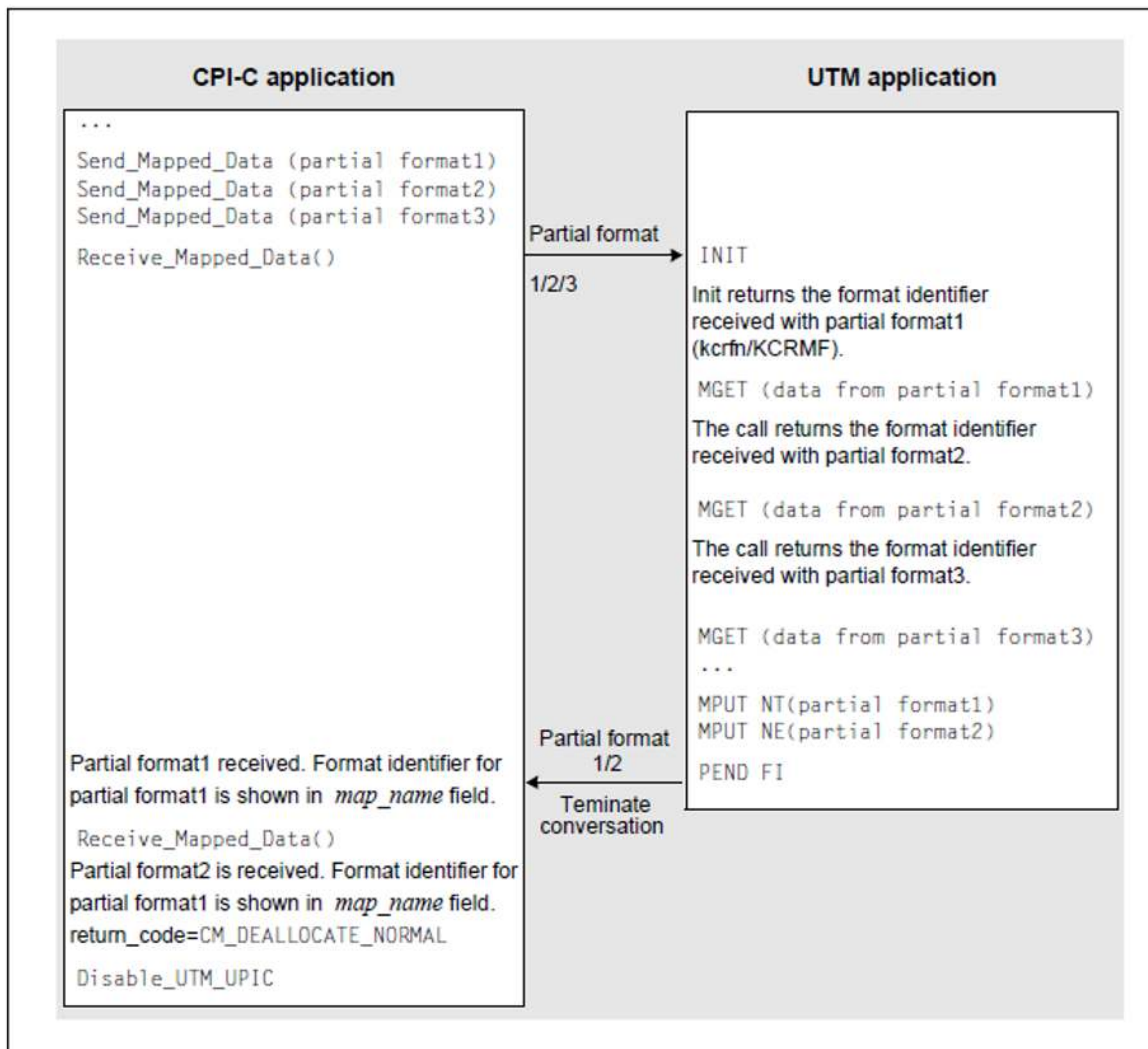


Figure 9: Exchange of formats

Detailed information on working with formats in a UTM application can be found in the openUTM manual „Programming Applications with KDCS".

UTM 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 configuring systems supported by openUTM. These format IDs consist of:

- a one-byte prefix specifying the type of the format (possible values are "*", "+", "#" and "−")
- 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
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".

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

3.3.4 UTM function keys

In an UTM application, function keys can be configured (F1, F2, ...F24 and in BS2000 systems also K1 through K14). Each function key can be assigned via UTM configuration a particular function, which openUTM executes when the function key is pressed.

A CPI-C client program can activate function keys in an UTM application.

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 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.

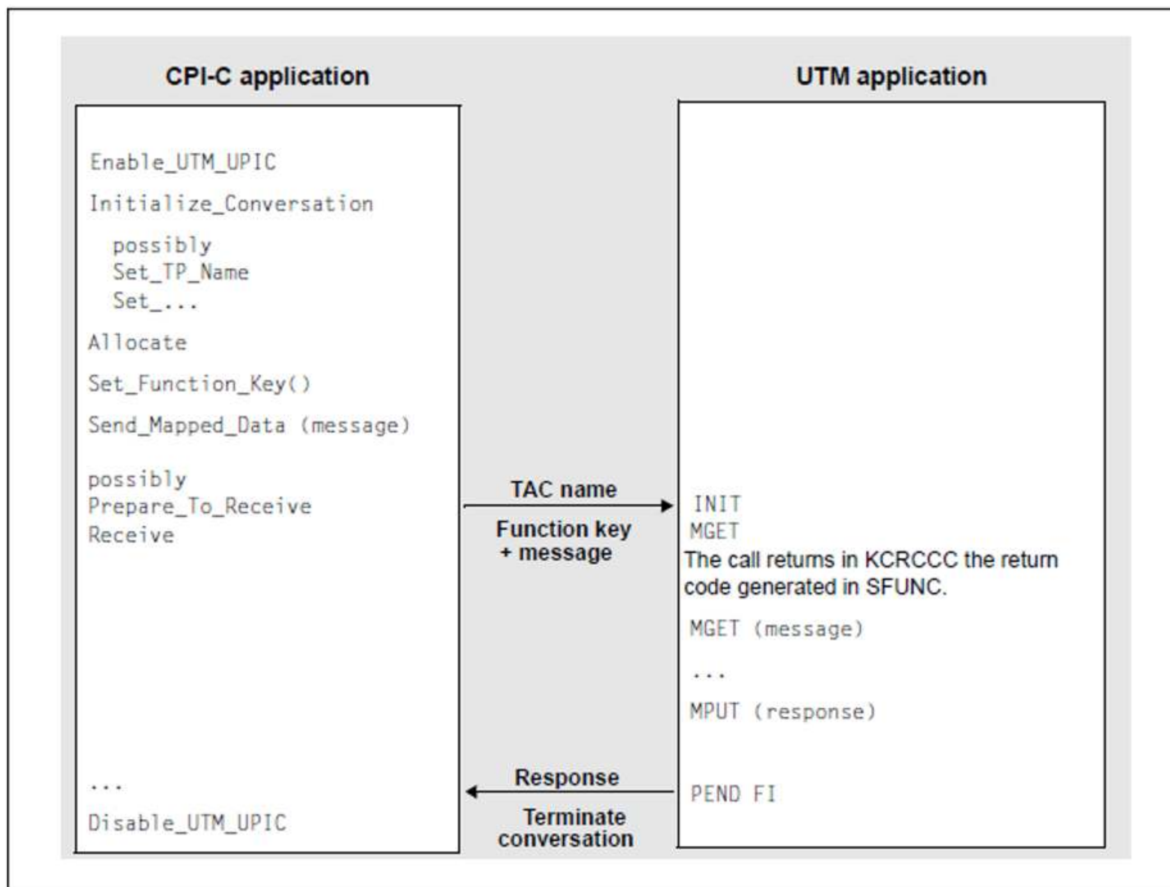


Figure 10: Pressing a function key in a UTM application

3.3.5 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.

3.3.6 Code conversion

With a heterogeneous link to a UTM application, it may be the case that different codes (ASCII, EBCDIC) are used in the client and the server systems, because Unix, Linux and Windows systems use ASCII compatible codes, while BS2000 systems use EBCDIC code, for example:

- a client application running on a Unix, Linux or a Windows system communicates with a UTM application on a BS2000 system.
- a client application running on a BS2000 system communicates with a UTM application on a Unix, Linux or Windows system.

In the case of such a heterogeneous link, messages which contain printable characters 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”). 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”).
- Automatic code conversion of the UPIC carrier system
  You activate automatic code conversion for the connection to a specific server using the *CHARACTER_CONVERTION* conversation characteristic. You can activate *CHARACTER_CONVERTION* 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”).
  - or by means of the *Set_Convertion()* 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, Linux or Windows systems based on the ASCII compatible code and with a UTM application on a BS2000 system based on an EBCDIC code (if the user data does not contain any binary information that was falsified during the code conversion).

> **CAUTION!**
>
> Keep in mind to convert the messages only once. Only messages containing printable characters may be safely converted. No conversion at all is allowed with a homogeneous link and with the link Windows system <-> Unix or Linux system.

3.3.6.1 Standard code conversion tables

The conversion tables are provided in a separate library.

At installation, the following files and libraries are installed:

Unix and Linux systems:

- *upic-dir*`/sys/libutmconvt.so` (conversion library)
- *upic-dir*`/kcsaeea.c` (source file for the conversion tables)

Windows systems:

In Windows, some of these files are installed as a 32-bit or 64-bit version depending on the platform and are given a corresponding suffix. This suffix (32 or 64) is indicated below as nn and is in italics.

- *upic-dir*`\sys\utmcnv`*nn*`.dll` (conversion library)
- *upic-dir*`\utmcnv\utmcnv`*nn*`.rc`, `resource.h` (resource files with version information)
- *upic-dir*`\utmcnv\kcsaeea.c` (source file for the conversion tables)

B2000:

- The conversion tables are located in the PLAM library $userid.SYSLIB.UTM-CLIENT.070 in the element KDCAEEA#LLM. This is also the location of the source file KDCAEEA.C.

Source file kcsaeea.c or KDCAEEA.C

The file `kcsaeea.c` or `KDCAEEA.C` contains eight tables for four code conversions. The tables provided convert the data as follows:

BS2000, Unix, and Linux systems:

- `kcsaebc` and `kcseasc`: ISO8859-i <-> EBCDIC.DF.04.i (EDF04i)
- `kcsaebc2` and `kcseasc2`: ISO8859-1 <-> EBCDIC.DF.04.DRV (EDF04DRV)
- `kcsaebc3` and `kcseasc3`: ISO646-IRV <-> EBCDIC.03.DF.03.IRV (EDF03IRV))
- `kcsaebc4` and `kcseasc4`: ISO646-IRV <-> EBCDIC.03.DF.03.DRV (EDF03DRV).

Windows systems:

- `kcsaebc` and `kcseasc`: Windows-1252 <-> EBCDIC.DF.04.F (EDF04F)
- `kcsaebc2` and `kcseasc2`: Windows-1252 <-> EBCDIC.DF.04.DRV (EDF04DRV)
- `kcsaebc3` and `kcseasc3`: ISO646-IRV <-> EBCDIC.03.DF.03.IRV (EDF03IRV))
- `kcsaebc4` and `kcseasc4`: ISO646-IRV <-> EBCDIC.03.DF.03.DRV (EDF03DRV).

In each case, the first and second code conversion are conversions between two 8-bit codes. The third and fourth code conversion are conversions between two 7-bit codes.

Adapting tables in kcsaeea.c or KDCAEEA.C

UPIC always uses the tables `kcsaebc` and `kcseasc` for the code conversions. If you want to modify the code conversion for your client applications, you have the following options:

- Modify the tables `kcsaebc` and `kcseasc` directly using the editor.

- Use another of the predefined code conversions (e.g. `kcsaebc2` and `kcseasc2`) and rename it to `kcsaebc` or `kcseasc`.
- Create your own tables and rename them to `kcsaebc` or `kcseasc`.

The following sections describe the individual steps necessary on the different platforms.

3.3.6.2 Modifying code conversion tables on Unix and Linux systems

In client applications on Unix and Linux systems, you can modify the standard conversion tables as follows:

1. Copy the file `kcsaeea.c` to a separate directory.
2. Modify the tables as required, see Paragraph AdaptingTables in Standard code conversion tables.
3. Compile the modified source file and use it to create a shared object.
4. Link the client application to this additionalEdit shared object.

3.3.6.3 Modifying code conversion tables on Windows systems

In client applications on Windows systems, you can modify the standard conversion tables as follows:

> **i** The version information of the created DLL is not essential in order to create the library.

Modifying the library utmcnv*nn*.dll

To modify the library `utmcnv`*nn*`.dll` the following steps are necessary:

1. Modify the tables as required, see “Adapting tables in kcsaeea.c or KDCAEEA.C (Standard code conversion tables)”.
2. Create the library `utmcnv`*nn*`.dll`.

   If you are using Microsoft Visual Studio:

   a. In the directory *upic-dir*`\utmcnv` create a new, blank Win32 project with the name `utmcnv64` (64-bit) and the application type *Dynamic-Link Library*.

   b. Add the following files to the project:

   - The modified code tables file `kcsaeea.c`,
   - If necessary, `utmcnv`*nn*`.rc`.

   c. Use this project to create `utmcnv`*nn*`.dll`.

Once the `utmcnv`*nn*`.dll` library has been created successfully, you still have to copy it into the *upic-dir*`\sys` directory containing the UPIC library `upicw`*nn*`.dll` or `upicws`*nn*`.dll` which is loaded by your application.

Verify that the original library `utmcnv`*nn*`.dll` is either overwritten by copying or is deleted, otherwise it may be loaded inadvertently by the system instead of the new library.

3.3.6.4 Modifying code conversion tables on BS2000 systems

In client applications on BS2000 systems, you can modify the standard conversion tables as follows:

1. Copy the file KDCAEEA.C to your user ID.
2. Modify the tables as required, see “Adapting tables in kcsaeea.c or KDCAEEA.C (Standard code conversion tables)”.
3. Compile the modified source file in LLM format to a PLAM library.
4. When starting your client application, use the SET- FILE-LINK command to assign a link name BLSLIB*nn* (with 00<=nn<=99) to the PLAM library with the LLM. Here, *nn* must be less than the number of the BLSLIB which you assign to the PLAM library $userid.SYSLIB.UTM-CLIENT.070. Alternative: Link the L element to your client application.

3.4 Communicating with an UTM application

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 Communicating in a multi-step UTM service with distributed transaction processing.

The calls used in the following examples are explained below:

- sign on to the UPIC carrier system (*Enable_UTM_UPIC*)
- initialize the conversation characteristics (*Initialize_Conversation*)
- establish the conversation (*Allocate*)
- send data (*Send_Data;* you can also use *Send_Mapped_Data*)
- receive the response (*Receive*; you can also use *Receive_Mapped_Data*)
- sign off from the UPIC carrier system (*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.

3.4.1 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

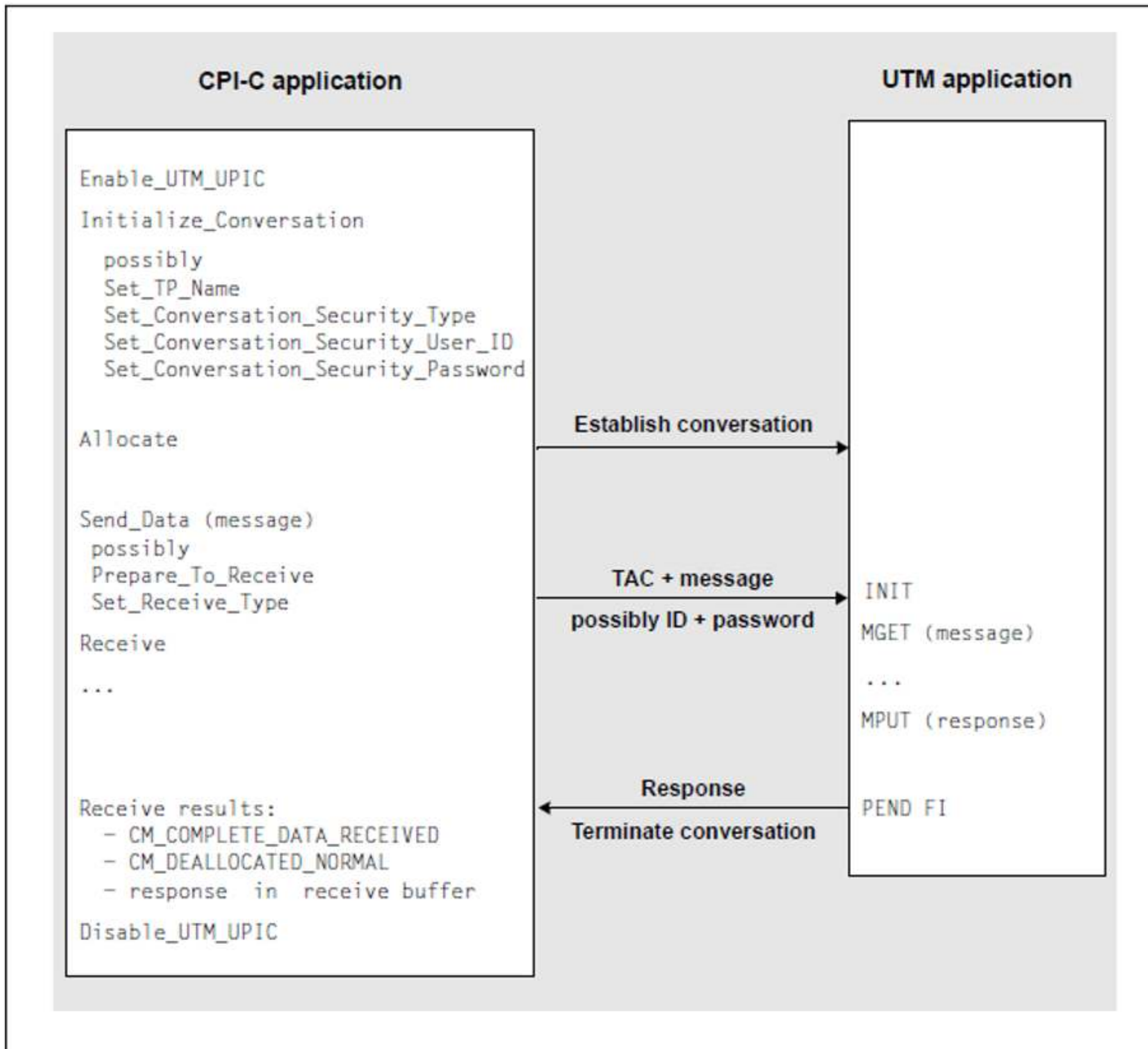


Figure 11: Single-step service with a *Send()*/*Receive()* call

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

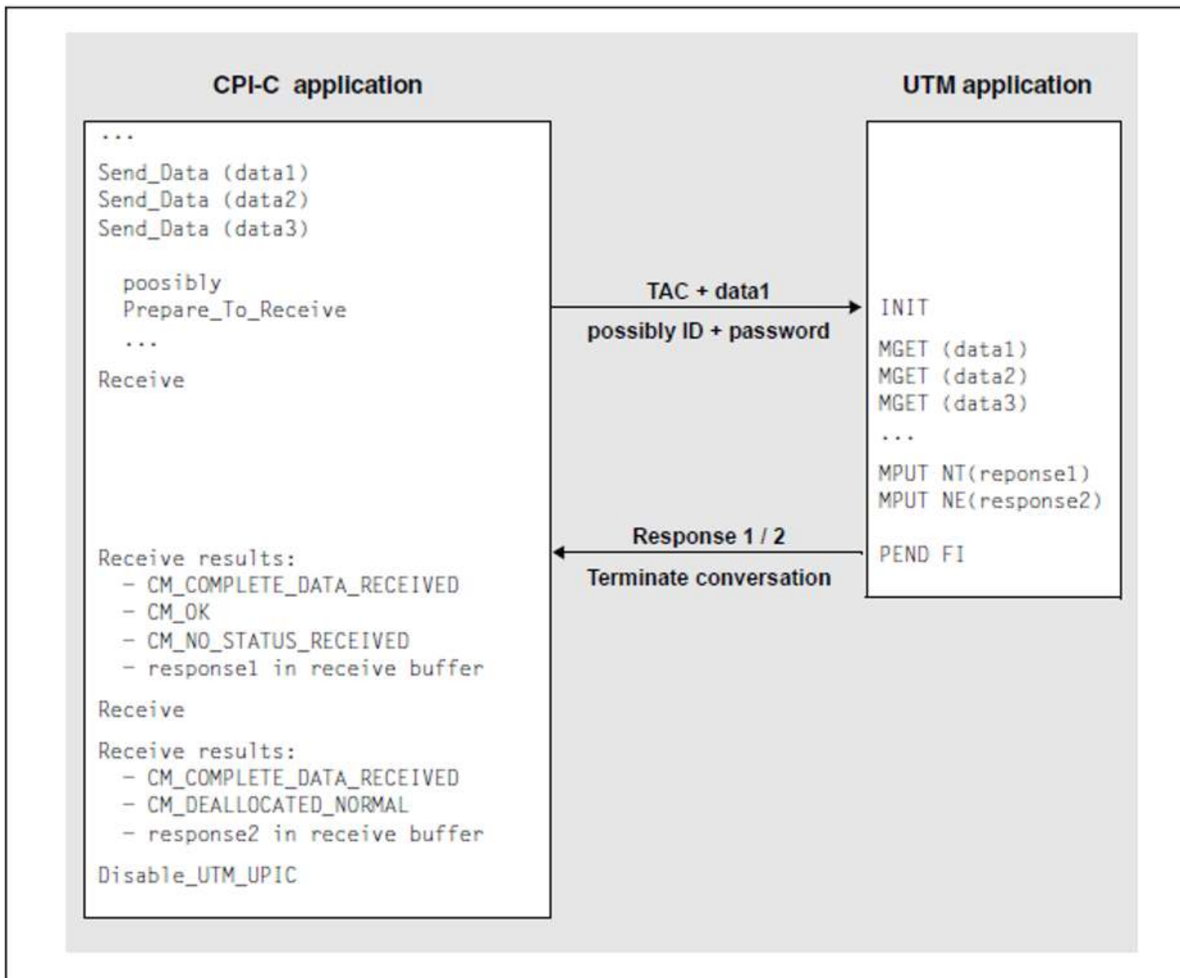


Figure 12: Single-step service with several Send/Receive calls

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 there are still more messages. Therefore, a second *Receive()* call is necessary to receive the second and last message. The last message is indicated by the return code CM_DEALLOCATED_NORMAL.

3.4.2 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.

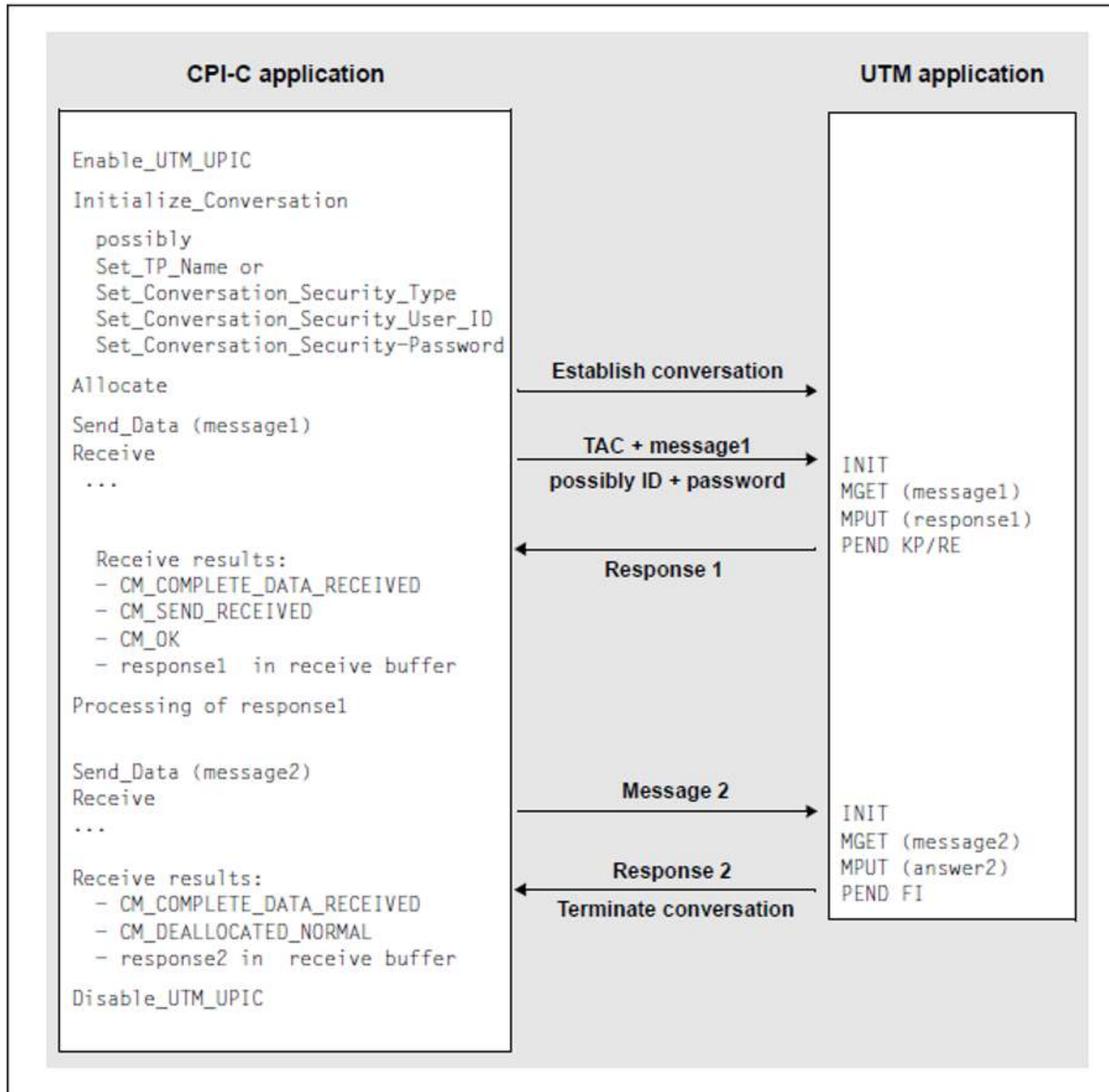


Figure 13: Multi-step service

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.

3.4.3 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.

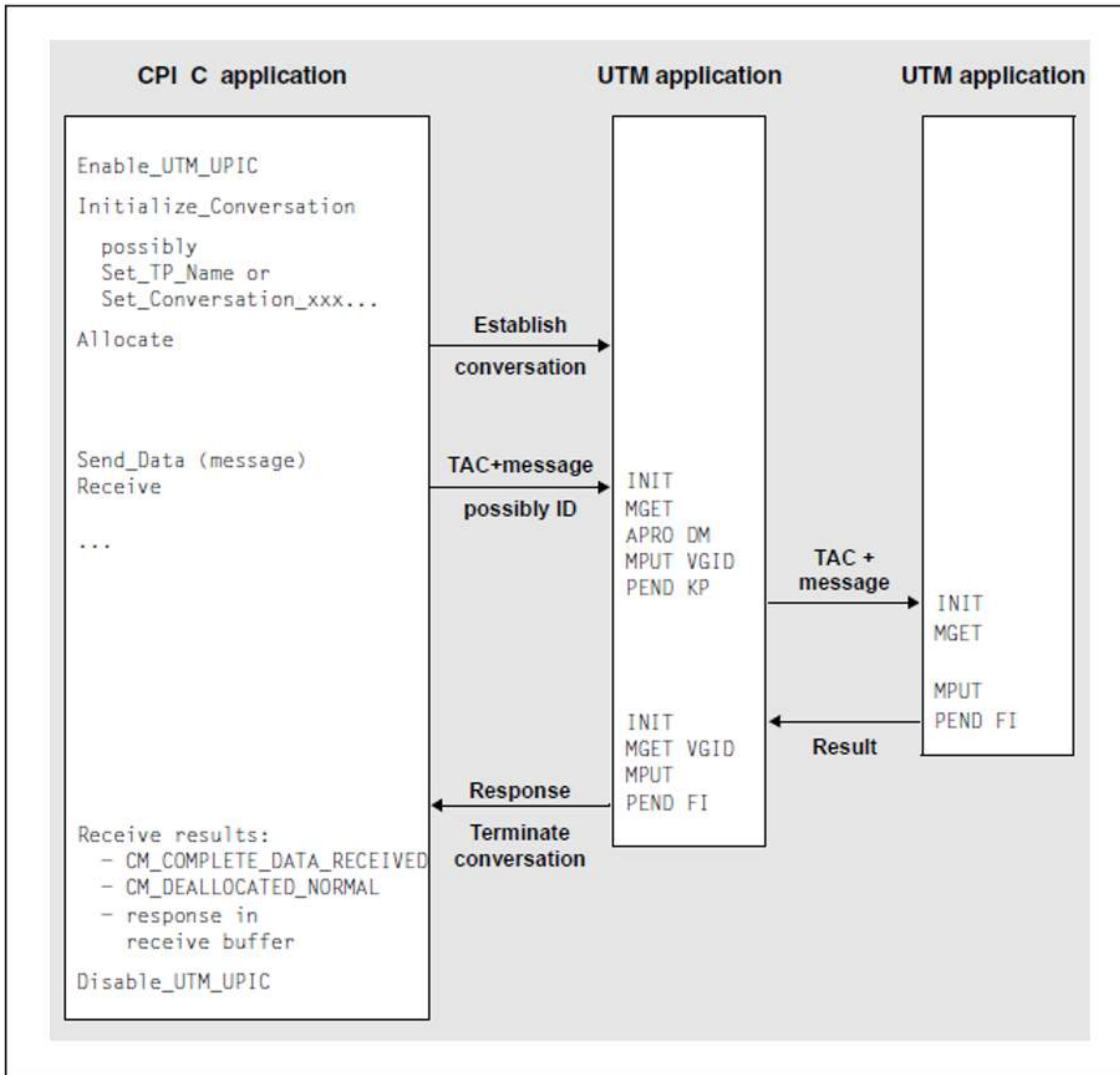


Figure 14: Multi-step service with DTP

3.4.4 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”. 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.

3.5 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.

3.5.1 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. In openUTM, 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 with 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 with a user ID (SIGNON ..., MULTI-SIGNON=YES), a UTM user ID is generated without restart (USER ...,RESTART=NO), then multiple sign-ons with this user ID are possible. The restartable service context is not required in this case.

3.5.2 Security functions

The following security functions are available in UTM:

- System access control functions

  These functions are available 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:

    Set_Conversation_Security_Type: define type of system access control
    Set_Conversation_Security_User_ID: specify UTM user ID
    Set_Conversation_Security_Password: specify associated password

  - In addition with UPIC

    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 *Receive* or *Receive_Mapped_Data* call:

    Extract_Secondary_Return_Code: query the secondary return code

  - On the XATMI interface, the *tpinit()* call has parameters to activate these system access control functions (see tpinit - Initializing the client.

  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 and Functions”).

  - By means of 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 configuration 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 (configuration 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 configuration. 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 configured with Grace Sign-On.
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 configured 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 configured for the UPIC client, the program is supplied with the return code CM_SECURITY_NOT_VALID after a *Receive* or *Receive_Mapped_Data* call. 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 configured 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 configured 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.

3.5.3 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" - XATMI. 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 Restart with Client Context.

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

- If a UTM user ID is configured as 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 configured as RESTART=NO, openUTM does not implement any service restarts, even if the LTERM partner used by the client is configured 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 reconfiguration.

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 application
- 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).

> **i** To avoid these problems, a UTM user ID configured as 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 re-configuraton of the UTM application, the program receives the return code CM_TP_NOT_AVAILABLE_NO_RETRY. openUTM then terminates the conversation.

> **i** The UTM utility KDCUPD transfers services of a client with restart capability.

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 configured 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

3.6 Encryption

Clients access unencrypted UTM services. 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 (man-in-the-middle attack). 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 hybrid encryption scheme. this is a combination of an asymmetric encryption for the exchange of the AES key and symmetric encryption with an AES key for the data.

How to enable encryption see section Runtime environment, linking, starting

Encryption methods

The encryption of user data for connections to UTM applications always follows the same pattern: First, a session key is exchanged or agreed between the two partners, and the user data is then encrypted with this session key between the two partners.

The session key used in all encryption levels is an AES key with a length of 128 bits.

For Encryption Levels 3 and 4, the session key is exchanged with the RSA algorithm. The client encrypts the created AES key with the public RSA key of the UTM application. Depending on the encryption level, an RSA key with a length of 1024 or 2048 bits is used.

At Encryption Level 5, the agreement of the AES key is done using the Elliptic Curve Diffie Hellman method. The RSA key of the UTM application is used in this method only to sign the public Diffie-Hellman key of the UTM application to prove the origin of the Diffie-Hellman key. The Diffie-Hellman method has the advantage that the AES key does not need to be transferred from the client to the server. Thus, this method offers Perfect Forward Secrecy.

Encryption levels 3 and 4 use the AES-CBC procedure to encrypt the user data. Encryption Level 5 uses the newer AES/GCM process. AES/GCM offers the advantage that in addition to the encryption of the user data, further protocol parts of the message are protected against changes by a message authentication code (MAC).

Configured encryption level	Public key	Symmetric key	authenticated encryption	perfect forward secrecy
TRUSTED	No key	No key	no	no
NONE	Depending on situation	Depending on situation	Depending on situation	Depending on situation
3	RSA - 1024 bit	AES (128 bit)	nein	nein
4	RSA - 2048 bit	AES (128 bit)	nein	nein
5	ECDH secp256k1	AES (128 bit)	ja	ja

Table 6: Configured 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. For encryption levels 3 and 4 the UPIC client can 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 and can be deleted by using calls of the UTM administration interface or by using the openUTM WinAdmin administration tool.

Requirements

If an encryption level of 3 to 5 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 of the encryption functionality is not available.

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 configured information for the partner.

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

ENCRYPTION-LEVEL=TRUSTED

The client is configured as trusted. In this case openUTM does not request encryption. Neither can the client force encryption.

ENCRYPTION-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.

The client generates an AES key. The client encrypts the AES 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 enforce 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 the software requirements for encryption are not met, passwords and user data are exchanged without encryption.

ENCRYPTION-LEVEL= 3 or 4

The UTM server sends the public RSA key associated with the appropriate encryption level. The length of this key is 1024 or 2048, see table6 Encryption.

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.

ENCRYPTION-LEVEL= 5

The UPIC client and the UTM application agree on a common secret with the ECDH procedure.

The client generates an AES key, encrypts it with the shared secret, and sends it back to the UTM application. openUTM saves 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, preferrably 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 2 or level 5 encryption is initialized, there are the following possibilities.

- *client-level* >= *tac-level*
  where the client has activated encryption for this conversation.
  The service can be started. The client sends user data in an encrypted form right from the beginning.
- *client-level* >= *tac-level*
  where the client has **not** activated encryption for this conversation 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.

3 ,4 or 5 is generated for the client

If a conversation whose TAC requires level 2 or level 5 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 terminates the conversation.

> i 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.

3.7 Multiple conversations (Unix, Linux and Windows systems)

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. Unix, Linux and Windows systems). For more information, see "Multithreading".

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 "Multiple conversations (Unix, Linux and Windows systems)").
- 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” capability.

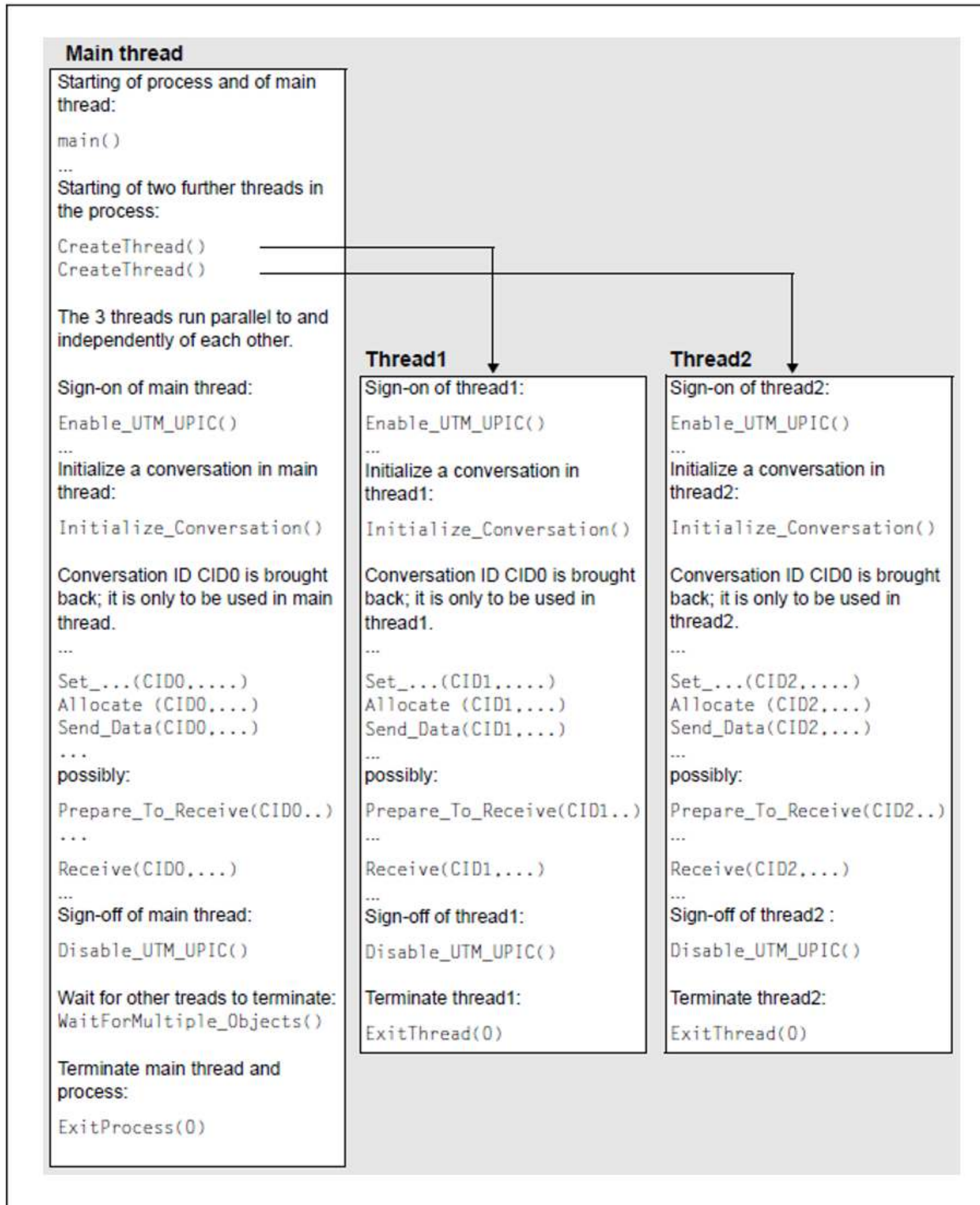


Figure 15: Starting several threads within a process (Unix, Linux and Windows systems) (the gray-hatched area corresponds to the process in which the client program is running)

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

Example of multiple conversations in Visual C++

```
void main ()                                                    1.
{
   ...
   thrd[0] = CreateThread(...,UpicThread,...);                  2.
```

```
   thrd[1] = CreateThread(...,UpicThread,...);
   ...
   Enable_UTM_UPIC (...);                                      3.
   ...
   /* Calls for establishing and processing a conversation          */
   /* in the main thread:                                          */
   Initialize_Conversation (...)
   ...
   Allocate (...)
   ....
   Send_Data (...)
   ...
   Receive (...)
   ...
   Disable_UTM_UPIC (...);
   ...
   WaitforMultipleObjects(2,&thrd[0],...);                     4.
   ExitProcess (0);                                            5.
}
DWORD WINAPI UpicThread(LPVOID arg)                            6.
{
   ...
   Enable_UTM_UPIC (...);
   ...
   /* Calls for establishing and processing conversation in thread  */
   /* as in main thread under 3.                                   */
   ...
   Disable_UTM_UPIC (...);
   ...
   ExitThread(0);                                              7.
}
```

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. *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 consecutively 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 consecutively. 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”).

3.8 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 "Side information for standalone UTM applications"). 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.

3.8.1 Multiple connections to the same UTM application with the same name

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

To enable a client to connect more than once with the same name, an LTERM pool which supports multiple connections with 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 connect to the UTM application (PTERM name), a PTERM statement must not be generated in the UTM application (see openUTM manual “Generating Applications”), otherwise multiple connections via the LTERM pool is not possible.

The CPI-C program can connect 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 connect.

3.9 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.

3.9.1 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” 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

Function	**Description**
Initialize_Conversation	Initializes conversation characteristic
Allocate	Starts a conversation
Deallocate	Ends a conversation abnormally
Send_Data	Sends data
Receive	Receives data

Table 7: 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

Function	**Description**
Convert_Incoming	Converts received data to the local code

Convert_Outgoing	Converts the data to be sent from the local code to the code of the communication partner
Deferred_Deallocate	Terminates the conversation as soon as the current transaction has been terminated successfully
Extract_Conversation_State	Inquires about the conversation state
Extract_Secondary_Information	Inquires about further information
Extract_Partner_LU_Name	Inquires about the value of the conversation characteristics *partner_LU_name* up to a maximum length of 32 bytes
Extract_Partner_LU_Name_Ex	Inquires about the value of the conversation characteristics *partner_LU_name* in full length
Prepare_To_Receive	Sends the data buffered in the send buffer to the communication partner immediately and switches to the “Receive” state
Receive_Mapped_Data [1]	Receives the data together with the structure information (format identifier)
Send_Mapped_Data [1]	Sends the data together with the structure information (format identifier)
Set_Conversation_Security_Password	Sets the password for a UTM user ID
Set_Conversation_Security_Type	Activates or deactivates the security function
Set_Conversation_Security_User_ID	Sets the UTM user ID
Set_Partner_LU_name	Sets the value for the conversation characteristics *partner_LU_name*
Set_Deallocate_Type	Sets values for the conversation characteristic *deallocate_type*
Set_Receive_Type	Sets values for the conversation characteristic *receive_type*
Set_Sync_Level	Sets values for the conversation characteristic *sync_level*
Set_TP_Name	Sets the name for a partner program (transaction code)

Table 8: Advanced Functions

[1]Not a component of X/Open CPI-C version 2

Additional UPIC functions

Function	Description
Enable_UTM_UPIC	Signs on to the UPIC carrier system
Extract_Client_Context	Outputs the client context
Extract_Conversation_Encryption_Level	Inquires about encryption level

Extract_Convertion	Queries the ASCII-EBCDIC conversion
Extract_Cursor_Offset	Inquires about cursor position offset
Extract_Max_Partner_Index	Queries the maximum index of the partner applications
Extract_Secondary_Return_Code	Queries secondary return codes
Extract_Shutdown_State	Queries the shutdown state of the server
Extract_Shutdown_Time	Queries the shutdown time of the server
Extract_Transaction_State	Queries the service and transaction state of the server
Disable_UTM_UPIC	Signs off from the UPIC carrier system
Set_Allocate_Timer	Setting timer for the Allocate call
Set_Client_Context	Sets the client context
Set_Convertion	Sets the ASCII-EBCDIC conversion
Set_Conversation_Encryption_Level	Sets encryption level
Set_Conversation_Security_New_Password	Sets a new password for a UTM user ID
Set_Function_Key	Sets the value of the function key to be transferred
Set_Partner_Index	Sets the index of the partner application
Set_Receive_Timer	Sets the timeout timer for the blocking receive of data
Set_Partner_Host_Name	Sets the host name of the partner application
Set_Partner_IP_Address	Sets the IP address of the partner application
Set_Partner_Port	Sets the TCP/IP port of the partner application
Set_Partner_Tsel	Sets the TSEL of the partner application
Set_Partner_Tsel_Format	Sets the TSEL format of the partner application
Specify_Local_Tsel	Sets the TSEL of the local application
Specify_Local_Tsel_Format	Sets the TSEL format of the local application
Specify_Local_Port	Sets the TCP/IP port of the local application
Specify_Secondary_Return_Code	Sets the properties of the secondary return code

Table 9: Additional UPIC Functions

3.9.2 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

only with UPIC local on Unix, Linux and Windows systems
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 by the UTM end because in the UTM application a TPOOL or PTERM connecting point is defined with ENCRYPTION_LEVEL=1 (or 3, 4, 5), but but the encryption requisites are not met.
- The transport connection was rejected by 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=2.

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.
- 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 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)
```

3.9.3 Convert_Incoming - Converting data from code of sender to local code

With the UPIC carrier system on Unix and Linux systems, the *Convert_Incoming* (CMCNVI) call converts the data form EBCDIC.DF.04.i to ISO8859-i by default.

With the UPIC carrier system on Windows systems, the *Convert_Incoming* (CMCNVI) call converts the data form EBCDIC.DF.04.F to Windows-1252 by default.

With the UPIC carrier system on BS2000 systems, *Convert_Incoming* (CMCNVI) converts the data from ISO8859-i to EBCDIC.DF.04.i.

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 (see chapter Code conversion) can be found at:
  - on Unix, Linux and Windows systems in the file `kcsaeea.c` under *upic-dir* or *upic-dir*`\utmcnv`.
  - on BS2000 in the file KDCAEEA.C in the library *$userid.*`SYSLIB.UTM-CLIENT.070`

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)
```

3.9.4 Convert_Outgoing - Converting data from local code to code of receiver

With the UPIC carrier system on Unix and Linux systems, the *Convert_Outgoing* (CMCNVO) call converts the data form ISO8859-i to EBCDIC.DF.04.i by default.

With the UPIC carrier system on Windows systems, the *Convert_Outgoing* (CMCNVO) call converts the data form Windows-1252 to EBCDIC.DF.04.F by default.

With the UPIC carrier system on BS2000 systems, *Convert_Outgoing* (CMCNVO) converts the data from EBCDIC.DF.04.i to ISO8859-i.

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 (see chapter Code conversion) can be found at:
  - Unix, Linux and Windows systems in the file `kcsaeea.c` under *upic-dir* or *upic-dir*`\utmcnv`.
  - on BS2000 systems in the file KDCAEEA.C in the library *$userid*.`SYSLIB.UTM-CLIENT.070`

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)
```

3.9.5 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)
```

3.9.6 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.

CMDFDE (conversation_ID, return_code)

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)
```

3.9.7 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”)
<-- 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)
```

3.9.8 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 CPI-C program and the UTM application (see also section “Initialize_Conversation - Initializing the conversation characteristics”).

In the `upicfile`, you can define a default name for the CPI-C application (LN.DEFAULT entry; see "Side information for the local application"). If the CPI-C program is to connect to the UPIC carrier system with this default name, it can specify an “empty local name” in the *local_name* field. UPIC then searches in the `upicfile` for the LN. DEFAULT entry and uses the corresponding local application name to establish the connection to the UTM application. Several CPI-C program runs can connect 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”):

with UPIC remote:

- Local application name defined in the `upicfile`.
- Name under which the program is known in CMX.
- Any name, whose properties 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 entry exists in the `upicfile` at the time of the call.

with UPIC local on Unix, Linux and Windows systems:

- 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, that is *local_name* will be treated as an "empty local 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 on Unix, Linux 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 connect to the UPIC carrier system simultaneously.

- A CPI-C program which has been started more than once can also connect 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 connect 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 on Unix, Linux and Windows systems* only: To enable the CPI-C program to connect 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 disconnect within the UTM application.

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 Unix, Linux and Windows systems* only*:* 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)
```

3.9.9 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, transferred to openUTM with *Set_Client_Context()* is saved until the end of the conversation unless it is overwritten with a new context. If the client requests a restart, the context is transferred back to the client together with the last dialog message, which openUTM has currently saved for this conversation.

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 return value of *data_received* is CM_NO_DATA_RECEIVED or the return 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. *The data_received* variable can have one of 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. CM_NO_DATA_RECEIVED No data was received.
<-- 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_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

  requested_length is less than the length of the internally buffered *client_context*, the buffer made available by the application program is completely filled and *data_received* is set to CM_INCOMPLETE_DATA_RECEIVED. If another CMECC call is then immediately made with a sufficiently large value for *requested_length* (i.e. >= 8), the buffer is read in full by such a 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 internal buffers. Check whether the memory requirement of your program is too high and if necessary reboot your system.

Function declaration: Extract_Client_Context

```
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 )
```

3.9.10 Extract_Conversation_Encryption_Level - Querying 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 on Unix, Linux and Windows systems: 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 *status_received* variable can have one of the following values: CM_ENC_LEVEL_NONE The user data of the conversation is transferred in unencrypted form. 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. CM_ENC_LEVEL_5 The user data is encrypted before transfer using the AES algorithm. For the exchange of the AES key, an ECDH algorithm with a key length of 2048 bits is used.
<-- return_code	Result of the function call.

Result (*return_code*)

CM_OK

The call is OK.

CM_CALL_NOT_SUPPORTED

only for *UPIC local on Unix, Linux and Windows systems*

The function is not supported. This return code 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 does not want encryption because the UPIC client is trusted.
- the UPIC client cannot implement encryption because the encryption functionality is not available.

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 return code only applies to Unix, Linux and Windows systems.
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

The encryption requirements are not met.

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

```
Extract_Conversation_Encryption_Level (unsigned char CM_PTR conversation_ID,
                                 CM_ENCRYPTION_LEVEL CM_PTR encryption_level,
                                 CM_RETURN_CODE      CM_PTR return_code )
```

3.9.11 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 *conversation_state* variable can have one of the following values: • 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 )
```

3.9.12 Extract_Convertion - Querying the value of the CHARACTER_CONVERTION conversation characteristic

The *Extract_Convertion* (CMECNV) call provides the program with the current value of the *CHARACTER_CONVERTION* 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_Convertion* call is permitted only in the “Initialize” 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_CONVERTION There is no automatic code conversion when data is sent or received. CM_IMPLICIT_CHARACTER_CONVERTION Data is automatically converted when sent or received (see also section “Code conversion”).
<-- 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_CONVERTION* 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_Convertion

```
CM_ENTRY Extract_Convertion(
                       unsigned char                CM_PTR  conversation_ID,
                       CM_CHARACTER_CONVERTION_TYPE  CM_PTR  convertion_type,
                       CM_RETURN_CODE               CM_PTR  return_code )
```

3.9.13 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_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 *IInitialize_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_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 )
```

3.9.14 Extract_Max_Partner_Index - Querying the maximum index of partner applications

Calling *Extract_Max_Partner_Index* (CMEPIN) provides the program with the number of partner applications in the partner applications list, i.e. the highest index set with *Set_Partner_Index()*

This function is one of the additional functions of the UPIC carrier system; it is not part of the CPI-C interface.

UPIC-Local on Unix, Linux and Windows Systems:

The call *Extract_Max_Partner_Index* is not supported for a connection using UPIC-L.

Syntax

```
CMEPIN (conversation_ID, partner_index, return_code)
```

Parameter

--> conversation_ID	Identification of the conversation
<-- partner_index	Returns the maximum index for a list of partner applications. Minimum: 1
<-- return_code	Result of the function call

Result (*return_code*)

CM_OK

Call ok

CM_CALL_NOT_SUPPORTED

This return code only applies to Unix, Linux and Windows systems.
The function is not supported. This return code only occurs with UPIC-L.

CM_PROGRAM_PARAMETER_CHECK

The value of the *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

The conversation is not in "Initialize” state.

CM_PRODUCT_SPECIFIC_ERROR

The UPIC instance could not be found or there is a memory bottleneck.

State change

The call does not change the state of the conversation.

Behavior in the event of errors

CM_CALL_NOT_SUPPORTED

This return code only applies to Unix, Linux and Windows systems.
Normal behavior if the application is linked to a UPIC-L library.

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.

Function declaration: Extract_Max_Partner_Index

```
CM_ENTRY Extract_Max_Partner_Index( unsigned char CM_PTR  conversation_ID,
                                   CM_INT32       CM_PTR  partner_index,
                                   CM_RETURN_CODE CM_PTR  return_code )
```

3.9.15 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.

! 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)
```

3.9.16 Extract_Partner_LU_Name_Ex - Querying full length partner_LU_Name

The call *Extract_Partner_LU_Name_Ex* (CMEPLNX) provides the program with the current *partner_LU_name* for the conversation at full length.

This call is one of the advanced functions.

Note

The call *Extract_Partner_LU_Name* returns names with a maximum length of 32 bytes.

Syntax

```
CMEPLNX(conversation_ID, partner_LU_name, requested_length,partner_LU_name_length,
return_code)
```

Parameter

--> conversation_ID	Identification of the conversation.
<-- partner_LU_name	Returns the *partner_LU_name*.
--> requested_length	Maximum length of *partner_LU_name* that can be received.
<-- partner_LU_name_length	Specifies the length of the value supplied in *partner_LU_name*. The value of *partner_LU_name_length* is undefined if the return code is different than CM_OK. Minimum: 1, Maximum: 73.
<-- return_code	Result of the function call.

Result (*return_code*)

CM_OK

Call OK

CM_PROGRAM_PARAMETER_CHECK

The value in *conversation_ID* is invalid or *requested_length* is not large enough to receive the *partner_LU_name*.

CM_PRODUCT_SPECIFIC_ERROR

The UPIC instance could not be found.

CM_PROGRAM_STATE_CHECK

The conversation is not in “Initialize” state.

State change

The call does not change the state of the conversation.

Note

- If the return code is different than CM_OK the value of *partner_LU_name* has no meaning.

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.

CM_PROGRAM_STATE_CHECK

Modify program.

Function declaration: Extract_Partner_LU_Name_Ex

```
CM_ENTRY Extract_Partner_LU_Name_Ex (unsigned char CM_PTR conversation_ID,
                                unsigned char  CM_PTR partner_LU_name,
                                CM_INT32       CM_PTR requested_length,
                                CM_INT32       CM_PTR partner_LU_name_length,
                                CM_RETURN_CODE CM_PTR return_code)
```

3.9.17 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 value of *data_received* is CM_NO_DATA_RECEIVED or the return value of *received_length* = 0, the content of *buffer* is 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.

CM_NO_DATA_RECEIVED

No data was received.

<-- 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 )
```

3.9.18 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. 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.
<-- secondary_return_code	Returns the secondary return code of the last CPI-C call. When *return_code* is not CM_OK, ithe 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 configureded.

CM_SECURITY_STA_OFF

The specified user ID is locked by configuration 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 configured with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG

The user is configured 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 configuredwith 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 configured 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 configuration.
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 cofigured 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. In this case, the secondary return code set by UPIC is CM_SECURITY_NEW_PSWORD_WRONG .

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.

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 *conversation_ID* is ignored if no conversation exists, i.e. the library is in the "Reset" state.
- If the *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 *Extract_Secondary_Return_Code* call.

- The program cannot use the call to obtain a secondary return code from a preceding *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 )
```

3.9.19 Extract_Shutdown_State - Querying the shutdown state of the server

By issuing the *Extract_Shutdown_State* (CMESHS) call, a program can obtain the current shutdown state of the UTM partner application.

The *Extract_Shutdown_State* call is permitted in the "Send" and "Receive" states as well as in the "Reset" state immediately after a *Receive-*/*Receive_Mapped_Data* call.

This function is not part of the CPI-C specification but an additional function of the UPIC carrier system.

Syntax

```
CMESHS (conversation_ID, shutdown_state, return_code)
```

Parameters

--> conversation_ID Identification of the conversation

<-- shutdown_state The value contains the shutdown state of the UTM partner application. The *shutdown_state* variable can have one of the following values:

- CM_SHUTDOWN_NONE:
  The application has not initiated a shutdown.
- CM_SHUTDOWN_WARN:
  The application has initiated SHUTDOWN WARN.
- CM_SHUTDOWN_GRACE:
  The application has initiated SHUTDOWN GRACE.

<-- return_code Result of the function call

Result (*return_code*)

CM_OK

Call OK

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 the end of the conversation or because no conversation existed at the time (there was no *Initialize_Conversation* call after the *Enable_UTM_UPIC* call).

CM_PRODUCT_SPECIFIC_ERROR

The UPIC instance could not be found.

State change

The call does not change the state of the conversation.

Note

- If the return code is different from CM_OK then the value of *shutdown_state* is of no significance.

- After the end of the conversation, the value of *conversation_ID* remains valid for this function call until *Initialize_Conversation* or *Extract_Shutdown_State* is called.

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_Shutdown_State

```
CM_ENTRY Extract_Shutdown_State(
                        unsigned char      CM_PTR  conversation_ID,
                        CM_SHUTDOWN_STATE  CM_PTR  shutdown_state,
                        CM_RETURN_CODE     CM_PTR  return_code )
```

3.9.20 Extract_Shutdown_Time - Query the shutdown time of the server

By issuing the *Extract_Shutdown_Time* (CMESHT) call, a program can obtain the current shutdown time of the UTM partner application.

The shutdown time is returned in printable format of length *received_length* and has the Universal Time Coordinated (UTC) time format. It still has to be converted to the time in the local time zone.

The *Extract_Shutdown_Time* call is permitted in the "Send" and "Receive" states as well as in the "Reset" state immediately after a *Receive*-/*Receive_Mapped_Data* call or after an *Extract_Shutdown_State* call .

This function is not part of the CPI-C specification but an additional function of the UPIC carrier system.

Syntax

```
CMESHT (conversation_ID, buffer, requested_length, data_received, received_length,
return_code)
```

Parameters

--> conversation_ID — Identification of the conversation

<-- buffer — Buffer in which the data is received.

If the return value of *data_received* is CM_NO_DATA_RECEIVED or the return value of *received_length* = 0, the content of *buffer* is undefined.

buffer returns the time at which the application is shut down. The individual bytes have the following meanings:

Bytes 1 - 8: Date in the format yyyymmdd:

- yyyy:Year, four-digit
- mm: Month
- dd: Day

Bytes 9 - 11

- ddd: Day in year

Bytes 12 - 17: Time in the format hhmmss (UTC format):

- hh: Hour
- mm: Minute
- ss: Second

--> requested_length — Maximum length of the data that can be received.

<-- data_received — Specifies whether the program has received all the data.

If the result (*return_code*) does not have one of the values CM_OK or CM_DEALLOCATED_NORMAL then the value of *data_received* is undefined.

The *data_received* variable can have the following values:

CM_COMPLETE_DATA_RECEIVED

The data was received in full.

CM_INCOMPLETE_DATA_RECEIVED

The data was not received in full.

CM_NO_DATA_RECEIVED

No data was received.

<-- received_length	Length of the received data. The value of *received_length* is undefined if *return_code* is not equal to CM_OK.
<-- return_code	Result of the function call

Result (*return_code*)

CM_OK

Call OK

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 the end of the conversation or because no conversation existed at the time (there was no *Initialize_Conversation* call after the *Enable_UTM_UPIC* call). Alternatively, the value for *requested_length* is greater than 32767 or smaller than 1.

CM_PRODUCT_SPECIFIC_ERROR

The UPIC instance could not be found.

Note

- This function has not been implemented at its full level of complexity in accordance with the CPI-C specification. The simplifications compared to CPI-C are as follows:

  The internal buffer possesses a restricted size of 1024 bytes.

  If the value of *requested_length* is smaller than the length of the internally stored extended information then the buffer made available by the application program is completely filled and *data_received* is set to CM_INCOMPLETE_DATA_RECEIVED. It is not possible to obtain the remaining data using further CMESHT calls.

- After the end of the conversation, the value of *conversation_ID* remains valid for this function call until *Initialize_Conversation* or *Extract_Shutdown_Time* is called.

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_Shutdown_Time

```
CM_ENTRY Extract_Shutdown_Time(
              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 )
```

3.9.21 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_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_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.

transaction_state (hexadecimal)	Meaning
17 08 dd dd 18 08 dd dd	End of the processing step; the transaction is not completed, the service is still open (PEND /PGWT KP).
15 06 dd dd 16 06 dd dd	End of the processing step; the transaction is completed, the service is still open (PGWT CM/PEND RE).
1A 04 dd dd	End of a service and end of the transaction (PEND FI).
30 04 dd dd	End of a service with memory dump (PEND ER).
31 04 dd dd	End of a service (system PEND ER, i.e. PEND ER by openUTM).
32 04 dd dd	End of a service due to abnormal task termination (only openUTM on BS2000 systems).
20 04 dd dd 21 04 dd dd	Roll back of the first transaction of a service and end of the service (PEND RS).
20 06 dd dd 21 06 dd dd	Roll back of a follow-up transaction to the last synchronization point; the service is still open (PEND RS).

For further information on PEND and PGWT calls refer to the openUTM manual „Programming Applications with KDCS".

Function declaration: Extract_Transaction_State

```
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 )
```

3.9.22 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, Linux and Windows systems, if the identifier HD is placed before the *symbolic destination name*
- in BS2000 systems, if the identifier SD is placed before the *symbolic destination name*.

For details see also "Side information for standalone UTM applications".

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 "Side information for standalone UTM applications") 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 or ND 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" (CPI-C terms). In all other error conditions, the program does not change its state.

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.
- Multiple programs can connect with 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, 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, Linux 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 BCMAP entry in BS2000 systems.
- Enter the current *sym_dest_name* in the `upicfile` or check the entry for *sym_dest_name* for correct syntax.
- With a local connection on Unix, Linux and Windows systems: 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 and ND entries, respectively.

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)
```

3.9.23 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-USERS=) 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.

CM_RESOURCE_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 )
```

3.9.24 Receive - Receiving data from a UTM service

A program uses the *Receive* (CMRCV) call to receive information from a UTM service.

The program must repeat the *Receive*-call until the return value of *return_code* is unequal CM_OK or *status_received*=CM_SEND_RECEIVED.

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 on Unix, Linux and Windows systems: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.
<-- data_received	Specifies whether the program has received data. If the result (*return_code*) is neither CM_OK nor CM_DEALLOCATED_NORMAL, the value of *data_received* is undefined. *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 (segment) available for the program was received. Permission to send may have been received.

CM_INCOMPLETE_DATA_RECEIVED

A message (segment) was not transferred in full to the program. If *data_received* has this value, the program must issue repeated *Receive* calls until the message (segment) is received in its entirety, i.e. until *data_received* has the value CM_COMPLETE_DATA_RECEIVED.

<-- 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.

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 *Send_Data* call.

Unless the return code is CM_OK, the value of *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

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 configured 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, 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 "Receive - Receiving data from a UTM service".

The secondary return codes can also be queried using the *Extract_Secondary_Return_Code* call, see "Extract_Secondary_Return_Code - Querying secondary return codes".

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 configured
  - 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 configured 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 configured 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 re-configured.

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 configuration 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 configured.

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 configured with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG

The user is configured 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 configured 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 transfered by KDCUPD does not satisfy the complexity or minimum length requirement defined in application configuration.

CM_SECURITY_TA_RECOVERY

A transaction restart is required for the specified user ID.

CM_SECURITY_PROTOCOL_CHANGED

The open sevice 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_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.

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_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* call, a program can only receive the amount of data specified in the *requested_length* parameter. It is therefore possible that a message (segment) is only partially received with the *Receive* call. The *data_received* parameter indicates as shown below whether a complete message (segment) available for the program was received:
  - If the program has already received the complete message (segment), the *data_received* parameter has the value CM_COMPLETE_DATA_RECEIVED.
  - If the program has not yet received all data of the message (segment), 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.
- 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.
- 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

CM_RESOURCE_FAILURE_RETRY

Re-establish conversation.

CM_RESOURCE_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 configured. Use a user ID that is configured or create or dynamically configure the user ID you want.

CM_SECURITY_STA_OFF

Configure 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 configured with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG

The user is configured with a chipcard.

CM_SECURITY_NO_RESOURCES

Try again later.

CM_SECURITY_NO_KERBEROS_SUPPORT

The user is configured with a Kerberos principal and cannot sign on via UPIC.

CM_SECURITY_TAC_KEY_MISSING

Configuration 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 configured complexity level, see KDCDEF statement USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT

Select a longer password or change configuration, 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_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

```
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 )
```

3.9.25 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 program must repeat the *Receive_Mapped_Data*-call until the return value of *return_code* is unequal CM_OK or *status_received*=CM_SEND_RECEIVED.

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.*
<-- 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.
<-- data_received	Specifies whether data was received in the conversation. *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 (segment) available for the program was received.

CM_INCOMPLETE_DATA_RECEIVED
A message (segment) was not transferred in full to the program. If *data_received* has this value, the program must issue repeated *Receive* or *Receive_Mapped_Data* calls until the message (segment) is received in its entirety, i.e. until *data_received* has the value CM_COMPLETE_DATA_RECEIVED.
The value of *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.

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 *Send_Data* call.

Unless the return code is CM_OK, the value of *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 configured 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, 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 "Receive_Mapped_Data - Receiving data and format identifier from a UTM service".

The secondary return codes can also be queried using the *Extract_Secondary_Return_Code* call, see "Extract_Secondary_Return_Code - Querying secondary return codes".

CM_TPN_NOT_RECOGNIZED

Possible causes:

- a service restart with KDCDISP was rejected as no UTM user ID configured 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 configured
  - 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 configured 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 configured 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 re-configured.

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 configuration 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.

Secondary return code (*secondary_return_code*)

CM_SECURITY_USER_UNKNOWN

The specified user ID is not configured.

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 configured with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG

The user is configured 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 configured 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 configuration.

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_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.

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_TP_NOT_AVAILABLE_RETRY/NO_RETRY
  CM_RESOURCE_FAILURE_RETRY/NO_RETRY
- 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 (segment) sent by the partner. The *data_received* parameter indicates as shown below whether there is still more message (segment) data to be read.
  - If the program has already received the complete message (segment), the *data_received* parameter has the value CM_COMPLETE_DATA_RECEIVED.
  - If the program has not yet received all data of the message (segment), 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 configured. Use a user ID that is configured or create or dynamically configure the user ID you want.

CM_SECURITY_STA_OFF

Configure 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 configured with a magnetic stripe card and cannot sign on via UPIC.

CM_SECURITY_CARD_INFO_WRONG

The user is configured with a chipcard.

CM_SECURITY_NO_RESOURCES

Try again later.

CM_SECURITY_NO_KERBEROS_SUPPORT

The user is configured with a Kerberos principal and cannot sign on via UPIC.

CM_SECURITY_TAC_KEY_MISSING

Configuration 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 configured complexity level, see KDCDEF statement USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT

Select a longer password or change configuration, 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 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

```
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 )
```

3.9.26 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

CM_RESOURCE_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 )
```

3.9.27 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; that is 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.
- 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

```
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 )
```

3.9.28 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 “Initialize” 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 on Unix, Linux and Windows systems: 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

This return code only applies to Unix, Linux and Windows systems.
The function is not supported. This return code only occurs for UPIC-L.

CM_PROGRAM_STATE_CHECK

The conversation is not in the “Initialize” 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 return code only applies to Unix, Linux and Windows systems.
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 excessive and if necessary reboot your system.

Function declaration: Set_Allocate_Timer

```
CM_ENTRY Set_Allocate_Timer ( unsigned char CM_PTR  conversation_ID,
                              CM_TIMEOUT     CM_PTR  allocate_timer,
                              CM_RETURN_CODE CM_PTR  return_code )
```

3.9.29 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_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_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 )
```

3.9.30 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 on Unix, Linux and Windows systems: 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_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. CM_ENC_LEVEL_5 User data are encrypted and authenticated, using the AES/GCM algorithm. The Diffie-Hellman algorithm is used to exchange the AES key with a length of 2048 bits.
<-- return_code	Result of the function call.

Result (***return_code*** **)**

CM_OK

The call is OK.

CM_CALL_NOT_SUPPORTED

This return code only applies to Unix, Linux and Windows systems.
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 “Initialize” 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 software requirements are not met.
- the UTM partner application does not want to implement encryption because the UPIC-L client is trusted.

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

This return code only applies to Unix, Linux and Windows systems.
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 loaded an old encryption library. 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 )
```

3.9.31 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 *ssecurity_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 password must consist of characters which are allowed in the UTM partner application, see openUTM manual “Generating Applications”, USER statement. 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: 16 If you specify 0 here, *security_new_password* is filled with 16 spaces, 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_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 16, 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 only blanks were specified for the new password, this means the UTM application should reset the password, that is 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

```
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 )
```

3.9.32 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”.
--> security_password_length	Length in bytes of the password specified in *security_password*.Minimum: 0, maximum: 16If you specify 0 here, *security_password* is filled with 16 blanks; that is no password is transferred to openUTM for checking access authorization.
<-- 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_password_length* is less than 0 or greater than 16.

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* characteristics 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”).
- 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 excessive 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 )
```

3.9.33 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

```
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 )
```

3.9.34 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 )
```

3.9.35 Set_Convertion - Setting the CHARACTER_CONVERTION conversation characteristic

The *Set_Convertion* (CMSCNV) call sets the *CHARACTER_CONVERTION* conversation characteristic.

Set_Convertion 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_Convertion* 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_convertion, return_code)
```

Parameters

--> conversation_ID	Conversation identifier
--> character convertion	Specifies whether code conversion for the user data is to be performed or not. The following values can be set for *character_convertion*: CM_NO_CHARACTER_CONVERTION There is no automatic code conversion when data is sent or received. CM_IMPLICIT_CHARACTER_CONVERTION Data is automatically converted when sent or received (see also section “Code conversion”).
<-- 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_Convertion

```
CM_ENTRY Set_Convertion(
                  unsigned char                  CM_PTR  conversation_ID,
                  CM_CHARACTER_CONVERTION_TYPE   CM_PTR  convertion_type,
                  CM_RETURN_CODE                 CM_PTR  return_code )
```

3.9.36 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 )
```

3.9.37 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, Linux and Windows systems support the function keys F1 through F20. openUTM on BS2000 systems supports the function keys K1 through K14 and F1 through F24. The value CM_UNMARKED specifies 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.

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, Linux 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)
```

3.9.38 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 on Unix, Linux and Windows systems:

The Set_Partner_Host_Name call is not supported for connection over UPIC-L.

UPIC-R using UTM 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:64
<-- return_code	Result of the function call

Result (*return_code*)

CM_OK

The call is OK

CM_CALL_NOT_SUPPORTED

This call is not supported 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.

CM_PROGRAM_PARAMETER_CHECK

The value of *conversation_ID* or *host_name_length* is invalid.

CM_PROGRAM_STATE_CHECK

The conversation is in the “Initialize” 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 return code only applies to Unix, Linux and Windows systems.
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 )
```

3.9.39 Set_Partner_Index - Setting the partner application index

The call *Set_Partner_Index* (CMSPIN) sets the index for the subsequent *Set_Partner_xxx* calls by the partner application in the conversation. It may no longer be called after the *Allocate* call. *Set_Partner_xxx* calls without a preceding *Set_Partner_Index* call are handled in the same way as after a *Set_Partner_Index* call with the index 1.

This function is one of the additional functions of the UPIC carrier system; it is not part of the CPI-C interface.

UPIC-Local on Unix, Linux and Windows systems:

The call *Set_Partner_Index* is not supported for the connection using UPIC-L.

Syntax

```
CMSPIN (conversation_ID, partner_index, return_code)
```

Parameter

--> conversation_ID	Identification of the conversation
--> partner_index	Specifies the *partner_index* to which the following *Set_Partner_xxx* calls relate. Minimum: 1 (default value); the sequence of *partner_index* values may not contain any gaps.
<-- return_code	Result of the function call

Result (*return_code*)

CM_OK

Call ok

CM_CALL_NOT_SUPPORTED

This return code only applies to Unix, Linux and Windows systems.The function is not supported.

For UPIC-L the return code always occurs.

For UPIC-R the return code only occurs if an openUTM cluster has been configured.

CM_PROGRAM_PARAMETER_CHECK

The value of the *conversation_ID* or for *partner_index* is invalid.

CM_PROGRAM_STATE_CHECK

The conversation is not in "Initialize” state.

CM_PRODUCT_SPECIFIC_ERROR

The UPIC instance could not be found or there is a memory bottleneck.

State change

The call does not change the state of the conversation.

Behavior in the event of errors

CM_CALL_NOT_SUPPORTED

Normal behavior if

- the application is linked to a UPIC-L library (on Unix, Linux and Windows systems),
- or an openUTM cluster has been configured.

In this case, the functionality is not available.

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.

Function declaration: Set_Partner_Index

```
CM_ENTRY Set_Partner_Index( unsigned char  CM_PTR  conversation_ID,
                            CM_INT32       CM_PTR  partner_index,
                            CM_RETURN_CODE CM_PTR  return_code )
```

3.9.40 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 on Unix, Linux and Windows systems:

The *Set_Partner_IP_Address*() call is not supported for connection over UPIC-L.

UPIC-R using UTM 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.

Result (*return_code*)

CM_OK

The call is OK.

CM_CALL_NOT_SUPPORTED

The function is not supported.

On Unix, Linux and Windows systems, 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.

The code is returned with UPIC-R for BS2000 systems 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 “Initialize” 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.
- *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

On Unix, Linux and Windows systems, 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 systems, 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 )
```

3.9.41 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 UTM 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: 73. 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 73.

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 to length 8 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 )
```

3.9.42 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 on Unix, Linux and Windows systems:

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: 1; maximum: 65535
<-- 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 systems:

- On Unix, Linux and Windows systems, 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 on BS2000 systems in the event that the UPIC library on the BS2000 system 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_PARAMETER_CHECK

The value of *conversation_ID* or *port_number* is invalid.

CM_PROGRAM_STATE_CHECK

The conversation is not in the “Initialize” 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, Linux 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 systems, 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 )
```

3.9.43 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 on Unix, Linux and Windows systems:

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

This return code only applies to Unix, Linux and Windows systems.
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 “Initialize” 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 return code only applies to Unix, Linux and Windows systems.
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 )
```

3.9.44 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 on Unix, Linux and Windows systems:

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

This return code only applies to Unix, Linux and Windows systems.
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 “Initialize” 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 return code only applies to Unix, Linux and Windows systems.
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

```
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 )
```

3.9.45 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 on Unix, Linux and Windows systems:
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 return code only applies to Unix, Linux and Windows systems.
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

```
CM_ENTRY Set_Receive_Timer ( unsigned char CM_PTR  conversation_ID,
                             CM_TIMEOUT     CM_PTR  timeout_time,
                             CM_RETURN_CODE CM_PTR  return_code )
```

3.9.46 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 on Unix, Linux and Windows systems:
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.
- 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 *cconversation_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 return code only applies to Unix, Linux and Windows systems.
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 )
```

3.9.47 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.

Note

The call serves only to improve the portability of CPI-C programs. Even if it returns CM_OK, *sync_level* is not changed. UPIC internally always uses "sync_level=CM_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_Sync_Level

```
CM_ENTRY Set_Sync_ Level ( unsigned char  CM_PTR  conversation_ID,
                           CM_SYNC_LEVEL  CM_PTR  sync_level,
                           CM_RETURN_CODE CM_PTR  return_code )
```

3.9.48 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 )
```

3.9.49 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 on Unix, Linux and Windows systems:

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: 1, maximum: 65535
<-- 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 systems.

On Unix, Linux and Windows systems, 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 on BS2000 systems in the event that the UPIC library on the BS2000 system 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.

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, Linux 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 systems, 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 )
```

3.9.50 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 a not a component of the CPI-C interface.

UPIC-Local on Unix, Linux and Windows systems:

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

This return code only applies to Unix, Linux and Windows systems.
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

This return code only applies to Unix, Linux and Windows systems.
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

```
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 )
```

3.9.51 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* ccharacteristic 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 on Unix, Linux and Windows systems:

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

This return code only applies to Unix, Linux and Windows systems.
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 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 *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

This return code only applies to Unix, Linux and Windows systems.
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 )
```

3.9.52 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

```
CM_ENTRY Specify_Secondary_Return_Code (
                           CM_INT32        CM_PTR  return_type,
                           CM_RETURN_CODE  CM_PTR  return_code )
```

3.10 COBOL interface

The CPI-C-COBOL program interface largely corresponds to the C interface described in section “CPI-C calls in UPIC”. 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

- on Window systems in the file *upic-dir*\\`copy-cobol` or *upic-dir*\\`netcobol` directory
- on Unix and Linux systems in the directory *upic-dir*`/copy-cobol85` or

  upic-dir`/copy-netcobol`
- on BS2000 systems in the library returned by the following SDF-P function

```
INSTALLATION-PATH(INSTALLATION-UNIT='UTM-CLIENT',LOGICAL-ID='SYSLIB',DEFAULT-PATH-
NAME='*UNKNOWN')
```

CMCOBOL must be included in the WORKING-STORAGE-SECTION using the COPY statement. The names of constants are derived 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-Statement to avoid the keyword TIMEOUT

```
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 on 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:

```
...
 WORKING-STORAGE-SECTION.
************************
 COPY CMCOBOL.
...
```

```
 PROCEDURE DIVISION.
*************************
...
 CALL "CMINIT" USING CONVERSATION-ID,SYM-DEST-NAME,CM-RETCODE.
```

4 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.

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.

4.1 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”.

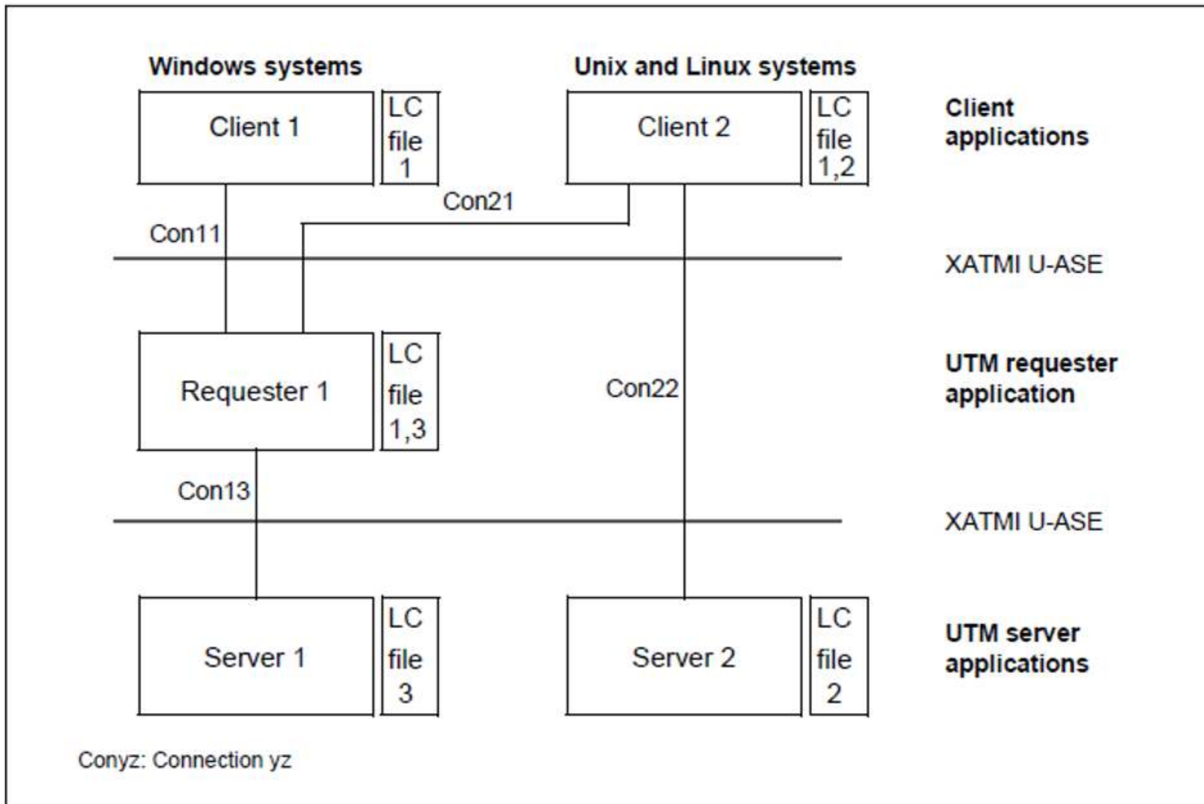


Figure 16: Client/server applications

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”).

4.1.1 Default server

To simplify the client/server configuration openUTM allows you to declare a default server using the statement DEST=.DEFAULT in the SVCU statement of the local configuration file (see "Creating the local configuration file").

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:

```
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
ND.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 the carrier system. However, if the local configuration file contains an entry `svcname2@BRANCH9`, this entry takes precedence over the default server assumption.

4.1.2 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.

4.2 Communication paradigms

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

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

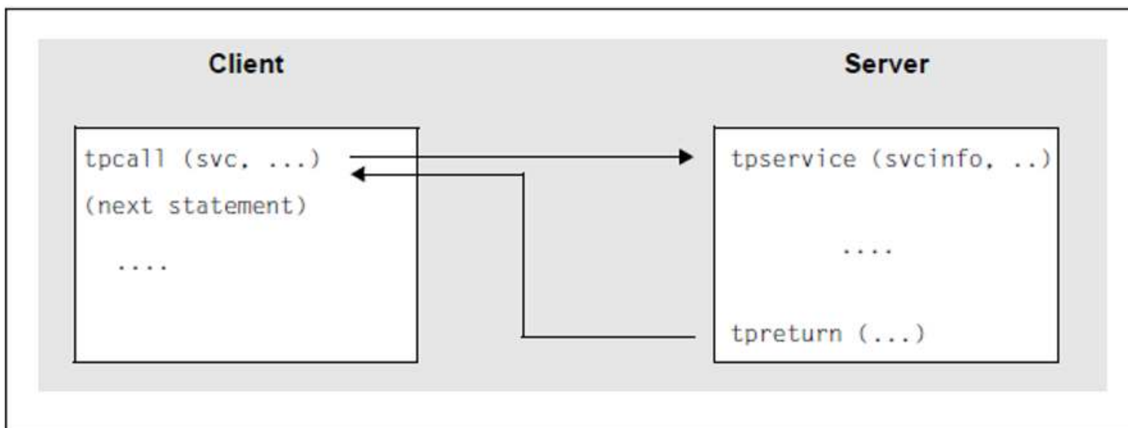


Figure 17: Synchronous request response paradigm

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.

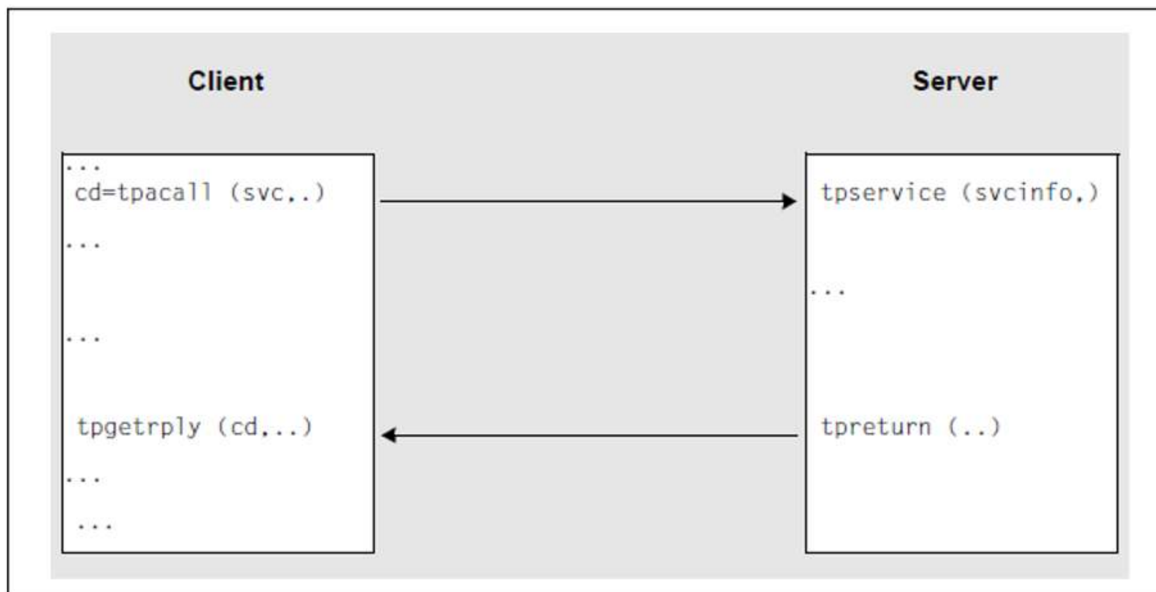


Figure 18: Asynchronous request response paradigm

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.

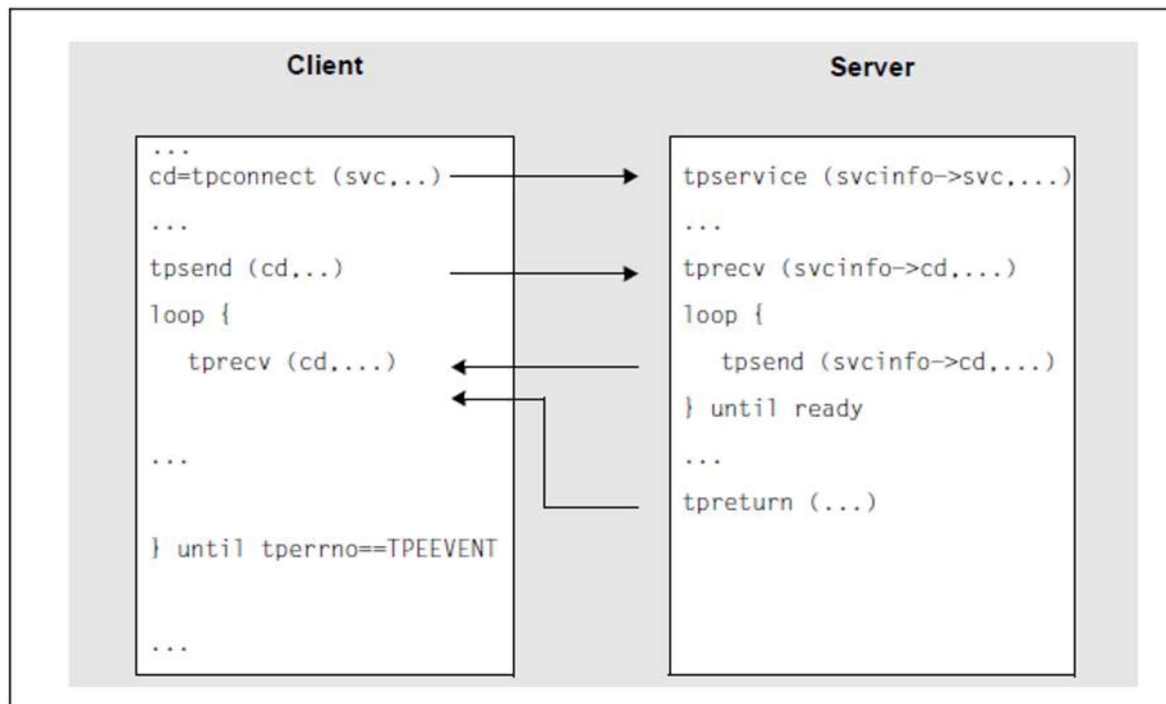


Figure 19: Conversational paradigm

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.

4.3 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 *tpalloc()* and are then called “typed buffers”; in COBOL programs, these buffers are defined statically and are then called “typed records”.

Types

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

Three types are defined in XATMI:

X_OCTET — Non-typed data stream of bytes (“user buffer”). This type has no subtypes.

No conversion takes place.

X_COMMON — All data types that can be used in common by C and COBOL.
Conversion is carried out by XATMI.

X_C_TYPE — All elementary C data types, with the exception of pointers.
Conversion is carried out by XATMI.

Subtypes

A subtype has a name of up to 16 characters, with which it is addressed in the application program. Each subtype is assigned a data structure (C structure or COBOL record) which determines the syntax of the subtype, see "Creating typed buffers".
The data types must not be nested.

The structure of a subtype is represented by a syntax string in the local configuration. In this string each elementary data type (basic type) is identified by a code which, if necessary, may also contain the field length specification (<m> and <n>).
The table below provides an overview of the elementary data types (basic types), their codes, and the character set of the string types:

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

T! <m>: <n>	character matrix, terminated by null ('\0')	SEQUENCE OF T.61-String	t61str[m][n]	--

[1]used in the local configuration to describe the data structures

[2]-- : not available in X_COMMON

[3]character matrix: two-dimensional character array

[4]in accordance with CCITT Recommendation T.61 or ISO 6937

The assignment between data structures, subtypes, and desired services is defined in the local configuration, see "Creating the local configuration file".

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`:

- This is **SD** or **ND** for Unix, Linux and Windows systems (stand-alone application).
- This is **HD** for BS2000 systems (stand-alone application).
- This is **CD** for the node applications of a UTM cluster application.

4.4 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.

4.4.1 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 *tpinit* and *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”.

Calls of the request/response paradigm

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

Table 10: Calls of the request/response paradigm

Calls for the conversational paradigm

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

Table 11: Calls for the conversational paradigm

Calls for typed buffers

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

Table 12: Calls for typed buffers

4.4.2 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:

Basic structure of a XATMI-Programm

```
tpinit()

XATMI-Aufrufe, z.B. tpalloc(), tpcall(), tpconnect(), ...tpdiscon()

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”.

4.4.2.1 tpinit - Initializing the client

Syntax in C

```
C:      #include <xatmi.h>
       int tpinit (TPCLTINFO *tpinfo)           (in)
```

Syntax in COBOL

```
COBOL: 01      TPINIT-REC.
               COPY TPCLTDEF.
       01      TPSTATUS-REC.
               COPY TPSTATUS.
       CALL "TPINIT" USING TPINIT-REC TPSTATUS-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.

The COBOL call needs two parameters:

- First parameter: TPCLTDEF record.
- The second parameter returns the return state of the call.

C structure TPCLTINFO

```
#define  MAXTIDENT  9
#define  MAXPASSWORD  17
typedef struct {
    long  flags;                    /* for future use */
    char  usrname[MAXTIDENT];
    char  cltname[MAXTIDENT];
    char  passwd [MAXPASSWORD];
} TPCLTINFO;
```

COBOL record TPCLTDEF

```
05 FLAG          PIC S9(9) COMP-5.
05 USRNAME       PIC X(8).
05 CLTNAME       PIC X(8).
05 PASSWD        PIC X(16).
```

A user ID is entered in *usrname* 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

- For Unix, Linux and Windows systems: 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 the BCMAP entry (BS2000 systems, see section "Configuration using BCMAP entries (BS2000 systems)").

If *usrname* and *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 *cltname* is initialized with the null string or with spaces, the local client name is preset to 8 spaces.

In C, if *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 *usrname*, *passwd*, and *cltname* (if any) must comply with the UTM name conventions, i.e. they can be up to eight (or 16 for *passwd*) characters in length, wherein the following applies:

- In C, they must be terminated with the end-of-string character ("\0").
- In COBOL, they must be filled up to the respective length if necessary.

Return codes

In the event of an error, *tpinit* returns in C the value -1 and sets the *tperrno* error variable to one of the following values:

TPEINVAL

One or more parameters were assigned invalid values.

TPENOENT

Initialization could not be performed, e.g. there may not be sufficient memory for internal buffers.

TPEPROTO

tpinit was called at an inappropriate time, e.g. the client is already initialized.

TPESYSTEM

An internal error has occurred.

In COBOL, in case of error in the TPINIT call the corresponding *tperrno* value is supplied directly as the return status.

4.4.2.2 tpterm - Signing the client off

Syntax in C

```
int tpterm ()
```

Syntax in COBOL

```
CALL "TPTERM" USING TPSTATUS-REC.
```

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 in C 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.

In COBOL, in case of error in the TPTERM call the corresponding *tperrno* value is supplied directly as the return status.

4.4.3 Transaction control

When an XATMI service is called, the client uses the call parameter *flag* (in C) or the TPTRAN-FLAG (in COBOL) to specify 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 rolled back.

> i 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.

4.4.4 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.

4.4.5 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 configured with administration authorization during KDCDEF configuration.

4.4.6 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 "Typed buffers".

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.

> i In Windows systems the XATMI interface is not supported in COBOL.

Windows systems

On Windows systems you will find the header files in the directory

upic-dir\xatmi\include

No COPY elements are supplied for COBOL.

Unix and Linux systems

On Unix and Linux systems you will find the header files in the directory

upic-dir/`xatmi/include`

and the COPY elements in the directory

upic-dir/`xatmi/copy-cobol85` or *upic-dir*/`xatmi/netcobol`

The openUTM client library is called `libxtclt` and is located in the directory *upic-dir*`/xatmi/sys`.

BS2000 systems

On BS2000 systems the include files and the COPY members are S type members of the library

$*userid*`.SYSLIB.UTM-CLIENT.070`

4.4.7 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 *tprevc* 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”.

4.4.8 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 "Typed buffers" are permitted.

The header files or COBOL COPY files in which the typed buffers are described serve as input for the configuration program `xatmigen`, see section “The xatmigen tool”. 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 "Typed buffers" are permitted. In particular, no pointer types are permitted in C.

The user may have to use the configuration tool `xatmigen` to map the character arrays to ASN.1 string types because neither C nor COBOL recognizes these data types; see section“The xatmigen tool”.

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

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

Example

C include for typed buffer

```
typedef struct {
        char   name[20];    /* person's name */
        int    age;         /* age */
        char   sex;
        long   shoesize;
} t_person;
struct t_city {
        char  name[32];             /* name of city */
        char  country;
        long  inhabitants;
        short churches[20];
        long  founded;
}
```

COBOL COPY for typed record

```
***** 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.

4.4.9 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.

4.5 Configuring

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

4.5.1 Creating the local configuration file

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

- A line begins with an SVCU 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 ("The xatmigen tool") 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.

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
ND.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.

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

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 configured 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=destination-name / .DEFAULT

A partner application identification of up to 8 bytes.

This name must be specified in the `upicfile` as the symbolic destination name (see "Configuring UPIC").

.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:

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

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 "Typed buffers".

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

During the configuration 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 "Typed buffers".

4.5.2 The xatmigen tool

The `xatmigen` tool 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:

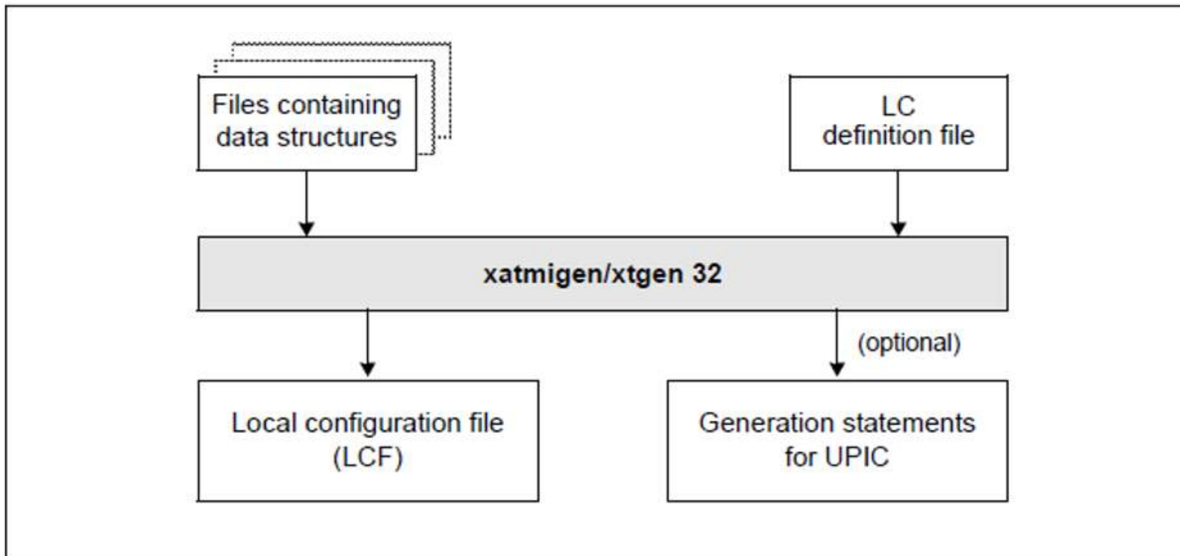


Figure 20: Working with xatmigen

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` configures 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 configuration statements for UPIC.

On Windows systems, success and error messages are written to the program window.

On Unix and Linux systems, success and error messages are written to `stdout` and `stderr`.

On BS2000 systems, 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

- On Windows systems `xatmigen` is called with

  `xatmigen[.exe] parameter`

  `xatmigen.exe` is located in the *upic-dir* `\xatmi\ex` directory.
- On Unix and Linux systems, `xatmigen` is called with

  `xatmigen` *parameter*

  `xatmigen` can be found in the *upic-dir* `/xatmi/ex` directory.
- On BS2000 systems, you start `xatmigen` with the following command:

```
/START-XATMIGEN
% CCM0001 PARAMETER EINGEBEN:
* parameter
```

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 option **-d** and, if specified, the options **-l** and **-c** must each be followed by the associated parameter. Specification of an option without a parameter is not permitted.

```
[ upic ]
 -d lcdf-name
[ -l lcf-name ]
[ -i ]
[ -c stringcode ]
[ descript-file-1 ] ... [ descript-file-n ]
```

upic
: If specified, a file `xtupic.def` containing entries for the configuration 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 configuration 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`".

  > i Any existing LCF of the same name is automatically overwritten.

  If the option 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`" option.

  The `-i` option takes priority over the `-c` option, if this is specified. If `xatmigen` is running in the background or in batch mode, the `-i` option 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`" option is ignored.

  The following values can be specified for *stringcode* (see table on "Typed buffers"):

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.

4.5.3 Configuring the carrier system and UTM partners

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

- with the UPIC carrier system, align the UPIC configuration (`upicfile`) with the local configuration and the partner configuration
- align the initialization parameters specified in *tpinit()* with the openUTM application configuration

4.5.3.1 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. For more information, please refer to section “Side information for stand-alone UTM applications” .

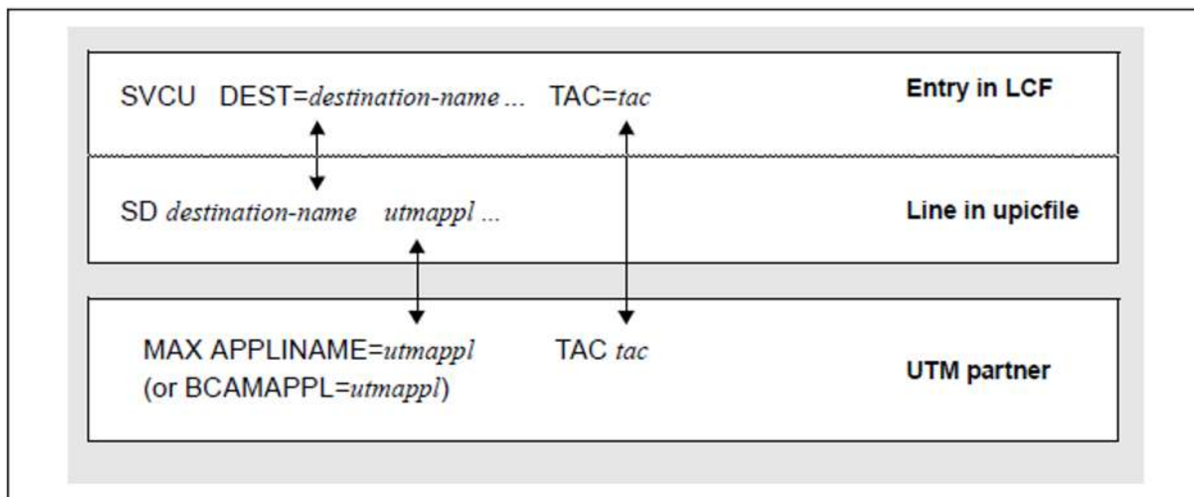


Figure 21: Conformance considerations when configuring server and client

Unix, Linux and Windows systems

An entry must start with **SD** or **ND** (Unix, Linux and Windows systems) if the server is a stand-alone application on a Unix or Windows system. If the server is a UTM cluster application then the entries for the node applications must start with CD, see section “Side information for UTM cluster applications”.

utmappl is the name of the UTM application, as configured with the KDCDEF statements MAX APPLINAME or BCAMAPPL=. Address information, such as IP address and port number, must be specified in the upicfile. With UTM applications on BS2000 systems, no UPIC communication can be made via the application name defined with MAX APPLINAME, since this is defined with T-PROT = NEA.

The transaction code *tac* in the SVCU statement must be defined with a TAC statement in the UTM configuration.

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.

4.5.3.2 Initialization parameters and UTM configuration

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 configuration as described below.

User ID and password

This security function can only be used with the UPIC carrier system.

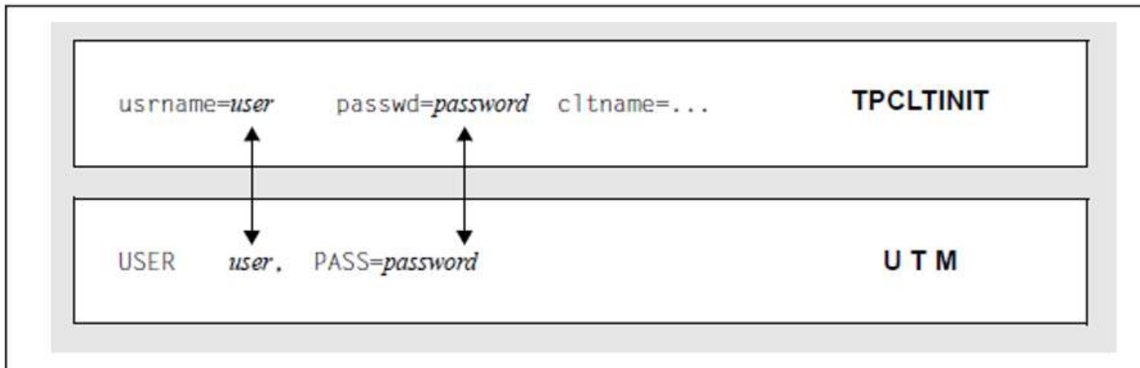


Figure 22: Corresponding configuration parameters

A corresponding UTM user ID must be configured in the UTM application with a USER statement for the user ID user passed with the *tpinit()* call. On the basis of the access data user and password, if given, UTM checks the access authorization.

Local name

The diagram below shows the initialization procedure in a case where a local application name is defined in the `upicfile`.

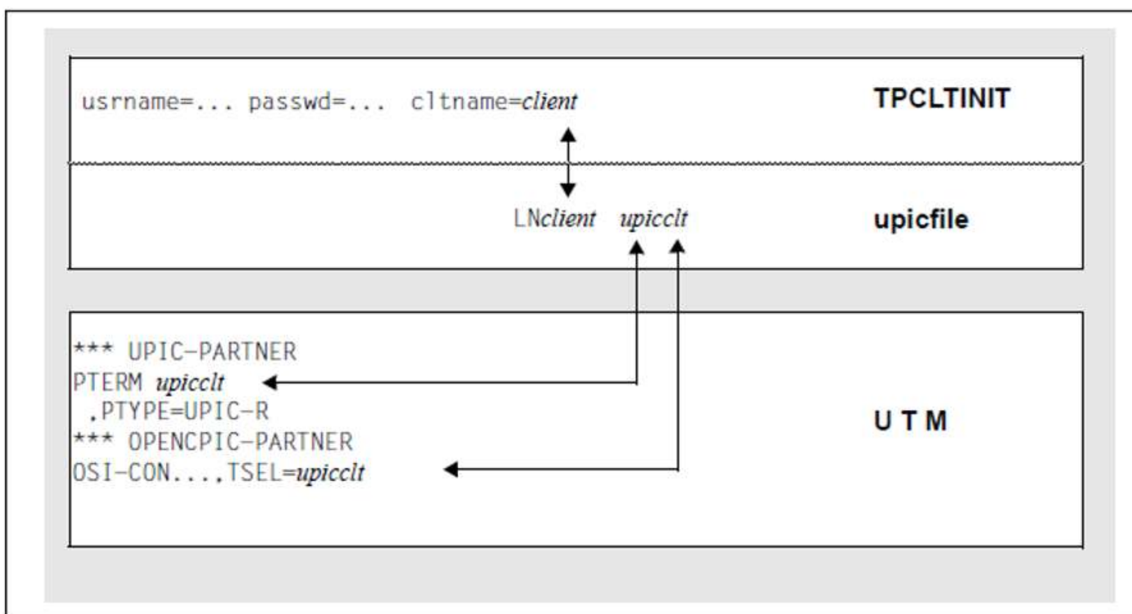


Figure 23: Initialization of a local application

If a local application name is configured 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

If no local application name is configured in the `upicfile`, the name defined on the UTM side in the PTERM statement 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 configuration.

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 (on BS2000 systems, also with parameter TPROT=ISO)
LTERM UPICTERM
PTERM UPRCLIENT, 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
```

4.6 Running XATMI applications

the following items are described in this section:

- Linking and starting an XATMI program
  - Linking an XATMI program on Windows systems
  - Linking an XATMI program on Unix and Linux systems
  - Linking an XATMI program on BS2000 systems
  - Starting the program
- Setting Environment variables on Unix, Linux and Windows systems
- Setting job variables on BS2000 systems
- Trace

4.6.1 Linking and starting an XATMI program

The following items are described in this section:

- Linking an XATMI program on Windows systems
- Linking an XATMI program on Unix and Linux systems
- Linking an XATMI program on BS2000 systems
- Starting the program

4.6.1.1 Linking an XATMI program on 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 `xtclt32.dll` or `xtclt64.dll` under *upic-dir*`\xatmi\sys`

The UPIC DLLs and the PCMX DLL must be available.

If you wish to run XATMI with UPIC-L on Windows, you must link the library `libxtclt.lib` into your application program.

4.6.1.2 Linking an XATMI program on Unix and Linux 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 and Linux systems)

Depending on whether UPIC-L or UPIC-R is used, the following XATMI and carrier-system libraries must be linked:

- UPIC local carrier system:
  1. `libxtclt` in the directory *utmpath*`/upicl/xatmi/sys`
  2. `libupicipc` in the directory *utmpath*`/upicl/sys`

  utmpath is the path name under which openUTM was installed.
- UPIC remote carrier system:
  1. `libxtclt` in the directory *upic-dir*`/xatmi/sys`
  2. CMX: `libxtclt` in the directory *upic-dir*`/xatmi/sys`
     Socket: `libupicsoc` in the directory *upic-dir*`/sys/`
  3. CMX library

4.6.1.3 Linking an XATMI program on BS2000 systems

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.070`

The library *$userid*.`SYSLIB.UTM-CLIENT.070` contains the example BIND-TPCALL for linking an XATMI program.

> i The link operation can be omitted if the link name BLSLIBxy is assigned to the required libraries in the correct order on program start.

4.6.1.4 Starting the program

An XATMI client program is started as an executable program.

4.6.2 Setting Environment variables on Unix, Linux and Windows 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 evaluatey by an XATMI application:

XTPATH 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 versions (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, Linux and Windows systems variable PATH: (*directory1* **:** *directory2* **:** ... or *directory1* `;` *directory2* `;` ...).

If the specified search path has more than 1024 characters the path is truncated!

There is a maximum of 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.

Setting environment variables on Windows systems

On Windows systems, you can set environment variables using the `Start/Settings/Control Panel`. You can then create or expand the environment variables here. On Windows systems, these settings remain valid until you change them again.

Setting environment variables on Unix and Linux systems

On Unix and Linux 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.

4.6.3 Setting job variables on BS2000 systems

Job variables can be set for an XATMI application. They are linked to the application using the following link names:

XTPATH	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.
XTLCF	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.
XTPALCF	Link to job variable containing the search path for additional descriptions of typecast buffers. The buffer descriptions are read from Local Configuration Files with the names XATMILCF or the name specified with XTLCF. The system continues to search for all the important XATMI configurations (e.g. SVCU ...) in the Local Configuration File specified by XTLCF. The system searches for Local Configuration Files under all the IDs specified in the search path and the descriptions of the typecast buffers are collected internally from these files (in the event of two identical names, only the first buffer description takes effect). The search path is specified in the form *userid1:userid2:...*.
XTCLTTR	Link to job variable containing the trace mode for the XATMI client application. Possible specifications:
	E (Error): Activates the error trace
	I (Interface): Activates the interface trace for the XATMI calls
	F (Full): Activates the full XATMI trace and that UPIC trace

Table 13: Job variables on BS2000 systems

If the software product JV is loaded as a subsystem, the job variables can be set as follows on BS2000 systems, 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=XTCLTTR, 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=XTCLTTR

The job variables are task-specific on BS2000 systems. Different job variables can be assigned to a second application running under the same ID.

4.6.4 Trace

Each client process writes the trace to a separate file, which can exist in two versions (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, Linux and Windows systems

  `XTC`*pid.n* ()

XTC	identifies an XATMI client trace
pid	process ID of the client process, 4 or 5-positions
n	number of the version: 1 or 2 the more recent trace can be identified by the time stamp

- BS2000 systems:

  `[prefix.]`XTC*tsn.n*

prefix	The part of the name specified in the job variable referred to by the linkname XTPATH (without terminating period).
XTC	identifies an XATMI client trace
tsn	ID of the client task, 4-digit
n	number of the version: 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

4.7 xatmigen messages

`xatmigen` messages have the form `XGnn messagetext...` and are output to *stderr* on Unix and Linux systems or to the program window on Windows systems and to SYSLST on BS2000 systems.

On Unix, Linux and Windows systems, use the LANG environment variable to specify whether you want German or English messages.

On BS2000 systems, 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 configuration fragment created
&CODE	string code for character array

`XG02 Generation terminated successfully`

Meaning
The LCF was created; configuration 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 configuration.

`XG04 Generation terminated by error`

`No file created.`

Meaning
The LCF was not created; the configuration 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 = configuration fragment file (=configuration 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: Basictype 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

5 Configuration

A client with the UPIC carrier system always uses UTM applications as servers in Unix, Linux or Windows systems or BS2000 systems. The configuration of the UPIC carrier system must therefore be coordinated with the configuration of the UTM partner application(s).

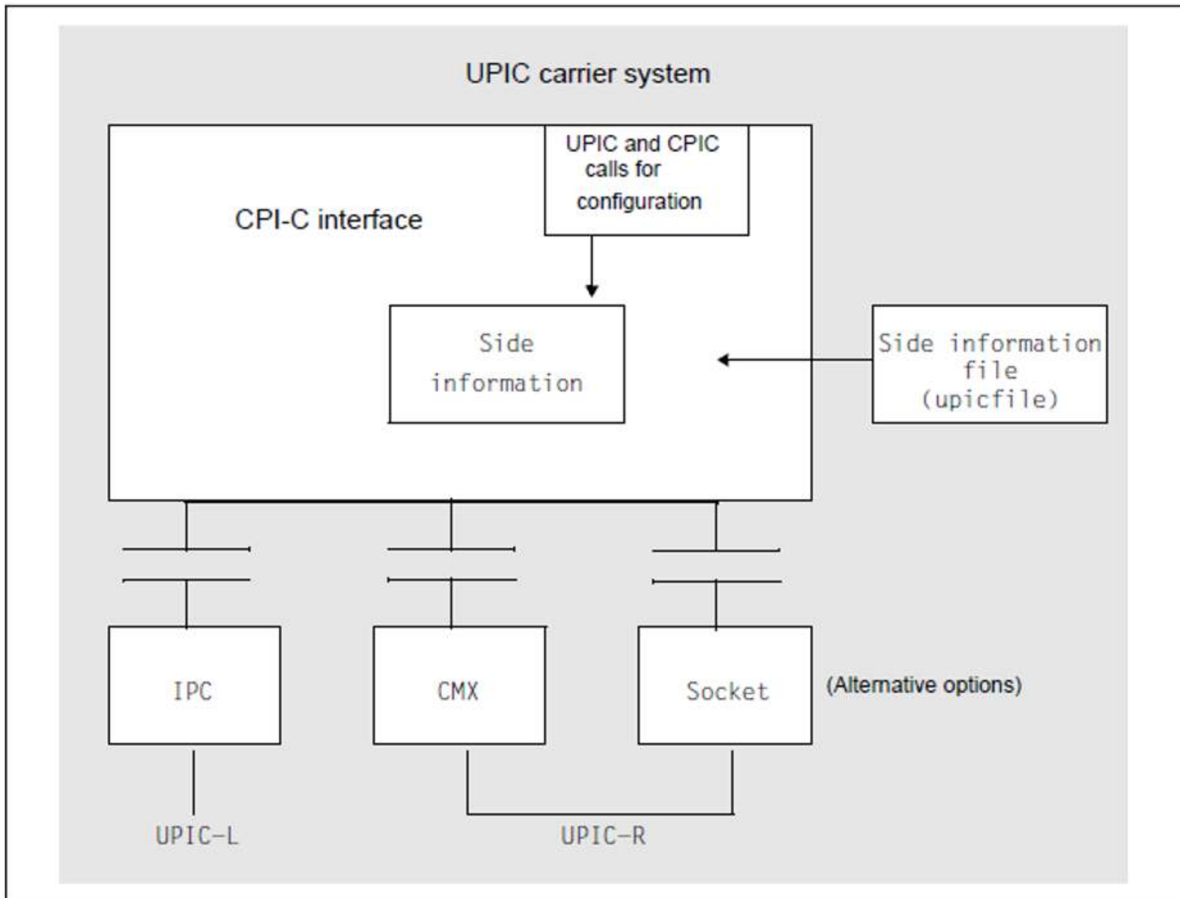


Figure 24: Configuration with and without side information file

5.1 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 in this case from BCAMAPPL and host name. UTM receives the local name of the client in order to be able to authenticate the client (PTERM statement).

openUTM only accepts the connection if a PTERM statement or a suitable TPOOL exists for the complete address consisting of local name, process name and BCAMAPPL.

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 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 preset *local_name* is assigned the following default value:

  - On Unix, Linux and Windows systems:
    - `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)”.

- *partner_LU_name*

  Following the *Initialize_Conversation* call, the *partner_LU_name* is assigned the following default value:

  - On Unix, Linux and Windows systems:
    - `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)”.

- *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)”.

5.1.1 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 systems, it has been agreed to use port number 102 wherever possible.
> There is no general recommendation with respect to the port number on Unix, Linux 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 systems 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.

> **BS2000 only**
>
> 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 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 and
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)”.

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)”.

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).

5.1.2 UPIC-L configuration (Unix, Linux and Windows systems)

UPIC-L uses the mechanisms of interprocess communication on Unix, Linux and Windows 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.

5.1.3 Configuration using BCMAP entries (BS2000 systems)

If UPIC uses the transport system component CMX(BS2000) for communication on BS2000 systems, 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 systems is to communicate with a UTM application on Windows systems.

5.2 The side information file (upicfile)

You must create the *upicfile* yourself. This file has the following format:

- In Unix, Linux and Windows 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.
- On BS2000: You must create a SAM file with the name `upicfile`. If you choose a different file name, you must set the job variable link *UPICFIL accordingly.

This file is used by all client programs, e.g. in the *Initialize_Conversation()* or *Enable_UTM_UPIC()* calls.

- On Linux Unix and Windows systems: The environment variable UPICPATH determines the directory; std = current directory
- On BS2000: the job variable linkname * UPICPAT determines a partially qualified file name; std = expiration identifier of the UPIC client

The `upicfile` can contain the following types of entries:

- communication partner entries which are addressed in the client program using the symbolic destination name:
  - Entries for the direct addressing of UTM applications (identifier HD or SD)
  - Entries for a list of communication partners (identifier ND) from which the client program selects an available UTM partner via the load balancer. These
    communication partners must be standalone UTM applications.
  - Entries for a list of communications partners in an openUTM cluster (identifier CD) from which the client program selects an available cluster node via the load balancer.
- Side information entries for the local application which are addressed in the client program using the local application name (identifier LN). 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.

5.2.1 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:

SD /HD /ND	symbolic destination name	blank	partner_LU_name	blank	transactioncode	blank	keywords	Z en lir char
2 bytes	8 bytes	1 byte	1-73 bytes[1]	1 byte	1-8 bytes	1 byte		
				--- optional ---		*--- optional ---*		

[1]For Unix, Linux and Windows systems: 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/ND and the symbolic destination name.
- Identifiers SD/HD/ND:
  The line begins with the identifier SD, HD or ND.

  The identifier HD or SD 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”.

  The identifier ND specifies that it is an entry for a list of partner applications. Please refer to section “Side information for list of partner applications” for details.
  HD and SD tags for Unix, Linux, and Windows systems:
  If you specify HD, then an automatic code conversion of the user data is carried out when sending and receiving.
  Data sent to the UTM partner application is converted from the locally used code to EBCDIC.
  Data arriving from the partner application is converted to local code by EBCDIC.
  Enter SD, then no automatic code conversion will be performed.
  Indicator HD and SD for BS2000 systems:
  On BS2000 systems, the tags have the opposite meaning.
  HD means in UPIC on BS2000 systems that no automatic code conversion is performed when sending and receiving data in the local system. HD should always be specified if the client communicates with a UTM application on BS2000 systems (BS2000 - BS2000 coupling).
  SD means that EBCDIC-> ASCII conversion is performed before sending data, and ASCII-> EBCDIC conversion when receiving.
  SD should be specified only for connections to UTM applications on Unix, Linux, or Windows systems.

  The indicator SD / HD in the upicfile can be overwritten with the *Set_Convertion()* 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 73 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*.

  The following restrictions apply for the name lengths:

  - *applicationname*: maximum length eight characters
  - *processorname*: maximum length 64 characters

  BS2000 systems
  You have to specify the *partner_LU_name* in two parts on BS2000 systems. *processorname* must then match the BCAM name of the remote computer.

  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 for Unix, Linux and Windows systems:

  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”) 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.

  ENCRYPTION-LEVEL={NONE | 0 | 3 | 4 | 5}
  ENCRYPTION-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=3, 4 or 5 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 3 to 5 mean:

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.

5 User data are encrypted and authenticated, using the AES/GCM algorithm. The Diffie-Hellman algorithm is used to exchange the AES key length of 2048 bits. Not available on BS2000

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:

- the software requirements are not met.
- it does not want to implement encryption because the client partner was generated as ’trusted’.

UPIC-L (Unix, Linux and Windows systems only): The value of ENCRYPTION-LEVEL is ignored.

The entry in the `upicfile` can be overwritten using the *Set_Conversation_Encryption_Level* call.

HOSTNAME=*hostname*

The host name is the processor name and can be up to 64 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.

UPIC-L (Unix, Linux and Windows systems only): 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 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. *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 (Unix, Linux and Windows systems only): The value for IP-ADDRESS is ignored.

UPIC on BS2000 systems 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 1 and 65535. The port number overwrites the port-number value assigned using *Initialize_Conversation*. Entering PORT is optional. 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 (Unix, Linux and Windows systems only): The value of PORT is ignored.

UPIC on BS2000 systems using CMX as its communication system: The value for PORT 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 (Unix, Linux and Windows systems only): 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 (Unix, Linux and Windows systems only): 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:

```
Valid formats for TSEL-FORMAT

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.

The value of TSEL-FORMAT is used. The entry in the `upicfile` can be overwritten using the *Set_Partner_Tsel_Format()* call.

UPIC-L (Unix, Linux and Windows systems only): 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 and Linux systems:*
    The line is concluded with a <newline> character (line feed). A semicolon before the <newline> character is optional.
  - *BS2000 systems:*
    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).

BS2000

In BS2000 systems, 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”). 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:

SD /HD /ND	.DEFAULT	blank	partner_LU_name	blank	transaction code	blank	keywords	end-of-line character
2 bytes		1 byte	1-73 bytes[1]	1 byte	1-8 bytes	1 byte		
				--- optional ---		*--- optional ---*		

[1]For Unix, Linux and Windows systems: With a local connection via UPIC locals, “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.

How to pass a list of communication endpoints via the upicfile is described in detail in the chapter Side information for UTM cluster applications.

5.2.2 Side information for list of partner applications

Each communication partner from the list of UTM partner applications is addressed using an identical symbolic destination name in the client program. This name is specified when initializing a conversation (call *Initialize_Conversation*). For each symbolic destination name used in the program, you must create entries in the `upicfile`.

To enable a UPIC client to access all communication partners, you create an entry in the `upicfile` for each partner. When doing this, please observe the following rules.

Rules for configuring a list of communication partners

- For a symbolic destination name, you must create a separate entry in the upicfile with the identifier ND for each partner application. For example, if the list consists of three UTM applications, you must create three entries with the same symbolic destination name.
- All entries for a particular symbolic destination name must immediately follow one another, see example below.
- The communication end points can belong to one specific UTM application or to different UTM applications. In this case, the UTM applications should be running on the same platform in order to avoid code conversion problems.

Example list of partner applications

You want to configure a list of three application names for a symbolic destination name (*service1*). The application names are distributed over two different standalone UTM applications running on the computers *HOST01* and *HOST02*. In the UTM application on *HOST01* the two application names (BCAMAPPL) *UTMAPPL1* and *UTMAPPL2* are configured, and in the UTM application on *HOST02* the application name *UTMAPPL1*.

The entries could, for instance, be as follows:

Sample for a list of partner applications

```
* entries for list of three communication end points in two UTM standalone applications
NDservice1 UTMAPPL1.HOST01 TAC1
NDservice1 UTMAPPL2.HOST01 TAC1
NDservice1 UTMAPPL1.HOST02 TAC1
```

5.2.3 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 UTM cluster application

- For each *symbolic destination name*, you must create a separate entry for each node application in the `upicfile` with the identifier 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.
- 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>
<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>
<tr><td rowspan="2">2 bytes</td><td rowspan="2">8 bytes</td><td rowspan="2">1 byte</td><td rowspan="2">1-73 bytes</td><td>1 byte</td><td>1-8 bytes</td><td>1 byte</td><td></td><td rowspan="2"></td></tr>
<tr><td colspan="2">--- optional ---</td><td colspan="2">--- optional ---</td></tr>
</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.
- 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 73 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*.

  The following restrictions apply for the name lengths:

  applicationname: maximum length eight characters

  processorname: maximum length 64 characters

BS2000 systems

On BS2000 systems, you must specify the *partner_LU_name* with two levels. *processorname* must then match the BCAM-name of the remote host.

Example: Specification in the upicfile

```
CDsymbdest 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" (CPI-C terms)) 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 | 3 | 4 | 5}:
  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= 3, 4 or 5 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 3 through 5 have the following meanings:

  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.

  5 User data are encrypted and authenticated, using the AES/GCM algorithm. The Diffie-Hellman algorithm is used to exchange the AES key with a key length of 2048 bits. Not available on BS2000.

  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

- the software requirements are not met
- 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 64 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 systems 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 1 through 65535. This port number overwrites the value for the port number assigned with *Initialize_Conversation*. The PORT specification is optional.

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 systems with CMX as the communication system

The value of PORT is ignored.

- 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:

Valid formats for TSEL-FORMAT

```
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.

The value of TSEL-FORMAT is used. The entry in the `upicfile` can be overwritten by a *Set_Partner_Tsel_Format* call.

- CONVERTION={IMPLICIT | NO}CONVERTION=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”.

  If you do not specify CONVERTION= or if you specify CONVERTION=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 and Linux systems:*
    Lines are terminated with a <newline> character (linefeed). A semicolon can be optionally inserted in front of the <newline> character.
  - *BS2000 systems:*
    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).

> **BS2000**
> Note that in BS2000 systems, 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 UTMAPPL2.

The entries could, for instance, be as follows:

Example of an upicfile

```
* entries for UTM cluster application UTMAPPL1
CDservice1 UTMAPPL1.CLNODE01 TAC1
CDservice1 UTMAPPL1.CLNODE02 TAC1
CDservice1 UTMAPPL1.CLNODE03 TAC1
* entry for stand-alone application UTMAPPL2
SDservice2 UTMAPPL2.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, HD or ND). 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"). 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:

CD	.DEFAULT	blank	partner_LU_name	blank	transaction code	blank	keywords	end-of line character
2 bytes		1 byte	1-73 bytes[1]	1 byte	1-8 bytes	1 byte		
				--- optional ---		*--- optional ---*		

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.

5.2.4 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:

<table>
<tr><th>LN</th><th>local application name</th><th>blank</th><th>application name</th><th>blank</th><th>keywords</th><th>end-of-line character</th></tr>
<tr><td rowspan="2">2 bytes</td><td rowspan="2">8 bytes</td><td rowspan="2">1 byte</td><td rowspan="2">1-32 bytes[1]</td><td>1 byte</td><td></td><td rowspan="2"></td></tr>
<tr><td colspan="2">--- optional ---</td></tr>
</table>

[1]For Unix, Linux and Windows systems: With local connection via UPIC local, “application name” can only be up to 8 bytes long.

Description of the entry

- 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 system using the application name.

  UPIC local (Unix, Linux and Windows systems only)*:* The application name can be up to 8 characters long.
- keywords (optional)
  The following keywords allow you to influence the UPIC-specific values for the local application (see also section “CPI-C terms”) 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 1 and 65535.
  >
  > 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 (Unix, Linux and Windows systems only): 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.
  >
  > The value of T-SEL is used. The entry in the `upicfile` can be overwritten using the *Set_Local_Tsel* call.
  >
  > UPIC-L (Unix, Linux and Windows systems only): 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:

Valid formats for TSEL-FORMAT
T for TRANSDATA
E for EBCDIC
A for ASCII

The use of T-SEL-FORMAT is optional.

The value of TSEL-FORMAT is used. The entry in the `upicfile` can be overwritten using the *Specify_Local_Tsel_Format* call.

UPIC-L (Unix, Linux and Windows systems only): 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 and Linux systems:*

    The lines are terminated with the <newline> character (linefeed). A semicolon can be optionally used before the <newline> character.
  - *BS2000 systems:*
    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”). 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>
<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>
<tr><td rowspan="2">2 bytes</td><td rowspan="2"></td><td rowspan="2">1 byte</td><td rowspan="2">1-32 bytes[1]</td><td>1 byte</td><td></td><td rowspan="2"></td></tr>
<tr><td colspan="2">--- optional ---</td></tr>
</table>

[1]For Unix, Linux and Windows systems: 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”).

5.3 Coordination with the partner configuration

BS2000 systems

If the client program is running on a BS2000 system, BCMAP entries may be required, see also "Configuration using BCMAP entries (BS2000 systems)".

There are dependencies between the entries in the client program, in the `upicfile` and the UTM configuration. The following sections describe which parameters you must coordinate for partner configuration.

You can specifiy 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:

Property	Function	Keyword	Default value
local application name			
T-SEL	Specify_Local_Tsel	T-SEL=	local application name
T-TSEL format	Specify_Local_Tsel_Format	T-SEL-FORMAT=	T
Port number	Specify_Local_Port	PORT=	102
transport address			
T-SEL	Set_Partner_Tsel	T-SEL=	partner name
T-TSEL format	Set_Partner_Tsel_Format	T-SEL-FORMAT=	T
Port number	Set_Partner_Port	PORT=	102
Internet address[1]	Set_Partner_IP_Address	IP-ADDRESS=	Information from host
Host name	Set_Partner_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 configuration of the UTM application.

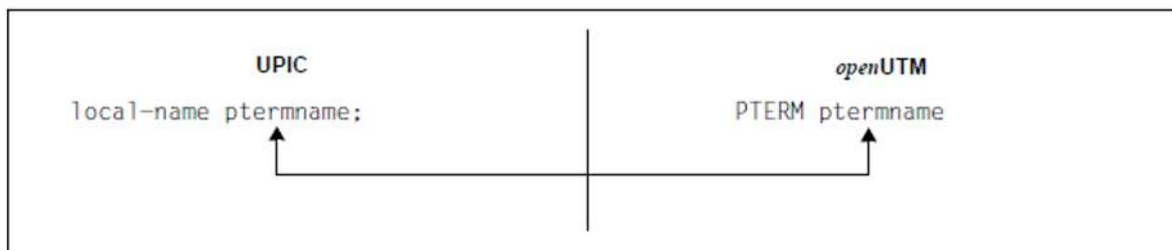
Local application name

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, Linux or Windows systems or on BS2000 systems without a BCMAP entry

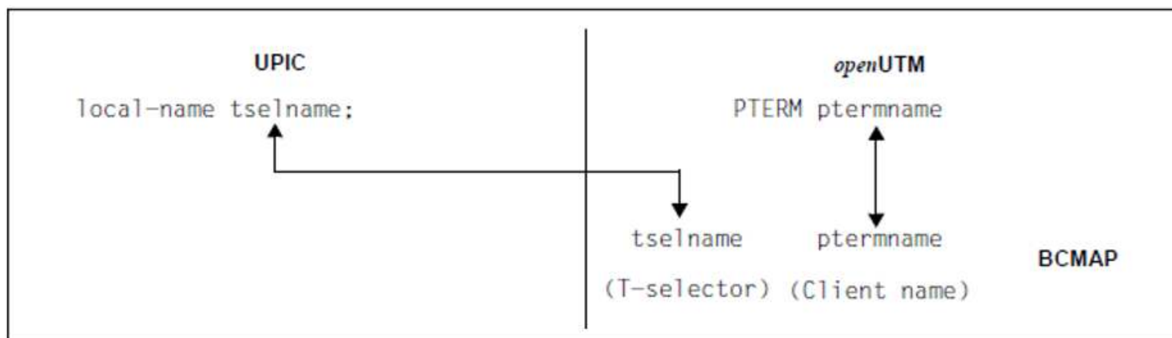
If the partner is a UTM application on a Unix, Linux or Windows system or a UTM application on a BS2000 system for which no BCMAP entries have been configured, the configuration must be coordinated as follows:



Both PTERM names must match. If there is no PTERM name configured for the client, there must be an LTERM pool via which the client can sign on.

Partners on BS2000 systems with a BCMAP entry

If the partner is a UTM application on BS2000 systems that uses BCMAP entries, the configurations must be harmonized as follows.



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* ("Side information for standalone UTM applications") is specified in two parts (*applicationname.processorname*), UPIC transfers this name directly to the transport system.

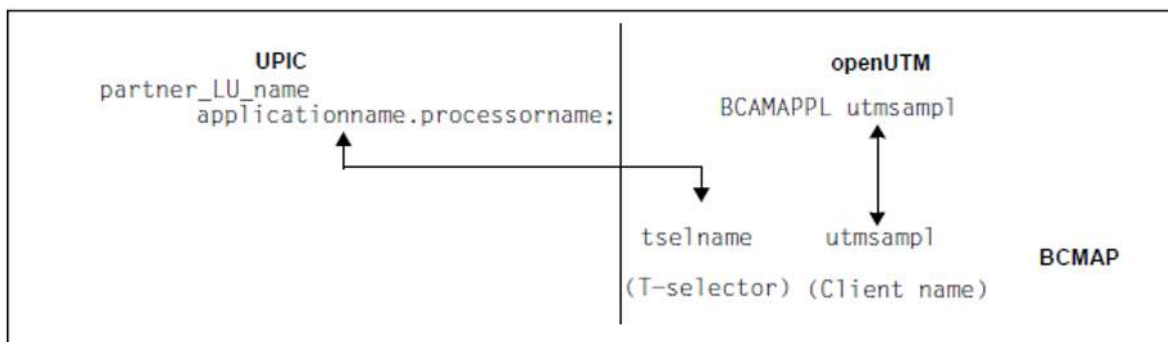
Partners on Unix, Linux or Windows systems or on BS2000 systems without a BCMAP entry

If the partner is a UTM application on a Unix, Linux or a Windows system or a UTM application on a BS2000 system for which no BCMAP entries have been configured, the configurations must be harmonized as follows:

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 systems with a BCMAP entry

If the partner is a UTM application on a BS2000 system that uses BCMAP entries, the configurations must be harmonized as follows.



applicationname must match the T-selector of the BCMAP entry for the UTM application on the remote processor.

6 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.

- Runtime environment, linking, starting
- Handling of CPI-C partners by openUTM
- Behavior in the event of errors
- Diagnostics

6.1 Runtime environment, linking, starting

Execution of CPI-C programs is controlled by environment variables or, on BS2000 systems, by the link name of the job variables. The following tables list the variables necessary for this.

Unix, Linux and Windows systems

Environment variable	Description
UPICPATH	Specifies the directory in which the side information file (*upicfile*) is stored. If the variable is not set, the file is sought in the current directory.
UPICFILE	Specifies the name of the side information file. If the variable is not set, the file name *upicfile* is set.
UPICLOG	Specifies the directory in which the log file is stored. The the default value set depends on the platform used; see section “UPIC log file”.
UPICTRACE	Controls the creation of a trace, see "UPIC trace".
UPIC_SSL_LIBRARY	specifies the name of the openSSL library. If the variable is not set, the following defaults are used: Unix- and Linux systems: `libssl.so` Windows systems: `libeay32.dll` If the the openSSL library can't be loaded, encryption functionality is not available.

BS2000 systems

Link name of the job variable	Description
UPICPAT	Specifies the partially qualified file name [:catid:$*progid.<partial-name>*] under which the side information file (*upicfile*) is stored. If the variable is not set, the system searches for the file under $*progid*. $*progid* is the user ID in which the program is running.
UPICFIL	Specifies the right-hand part of the name of the side information file. If this variable is not set, the file name is set to upicfile. The complete file name is composed of UPICPAT.UPICFIL.If neither UPICPAT nor UPICFIL is set, the file name is "$*progid*.UPICFILE".
UPICLOG	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 UPIC trace.
UPICTRA	Controls configuration of a trace, see section “UPIC trace”.

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.

- Implementing on Windows systems
- Implementation on Unix and Linux systems
- Using on BS2000 systems

6.1.1 Implementing on Windows systems

When creating and implementing CPI-C applications, take into account the special features described in section Compilation, linking, starting on Windows systems and in section “Runtime environment, environment variables on Windows systems”.

When creating and implementing UPIC-local applications on Windows systems, you must also take into account the specifications described in section “Special features implementing UPIC local on Windows systems”.

> i The setup for the UPIC client on Windows systems contains both the 32-bit and 64-bit variant. During the installation operation, the appropriate variant is installed depending on the system architecture or the selection..
> In the case of PCMX (Windows), there is a separate package for 32-bit and 64-bit environments. i.e, it is necessary to install the required PCMX packages depending on the UPIC bit mode.

6.1.1.1 Compilation, linking, starting on Windows systems

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`.

  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`
  - `UTM_ON_WIN32` on 32-bit platforms
  - `UTM_ON_WIN32` **and `UTM_ON_WIN64`** on 64-bit platforms

  You can see the effect of these options in the header file `upic.h`. It is located in the directory *upic-dir*`\include`.

- 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` (32-bit) or `upicw64.lib` (64-bit), located in the *upic-dir*`\sys` directory.
  - For programs which want to use Socket interface:
    the library `upicws32.lib` (32-bit) or `upicws64.lib` (64-bit), 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.

6.1.1.2 Runtime environment, environment variables on Windows systems

The environment variables listed in the table on "Runtime environment, linking, starting" are used for controlling CPI-C applications.

The path name can be given with spaces in the UPICTRACE variable. If spaces are used, then the path name must be enclosed in double quotes. Double quotes can also be used if there are no spaces in the path name.

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).

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.

6.1.1.3 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* `\upicl\sys\libupicl.lib`, which must be linked to every client program and, if necessary,
- *utmpath* `\xatmi\sys\libxtclt.lib`, which must also be linked to XATMI programs.

For further information on *utmpath*, refer to openUTM manual "Using UTM Applications on Unix, Linux 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. The PATH environment variable must be manually extended as required following the installation openUTM.

Configuring a UPIC-local client with Visual C++ Developer Studio

The following briefly describes how you can configure a UPIC-local client project using the Visual Studio.

> i 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 Studio. The *Project Settings* dialog box is displayed on the screen. Now proceed as follows:

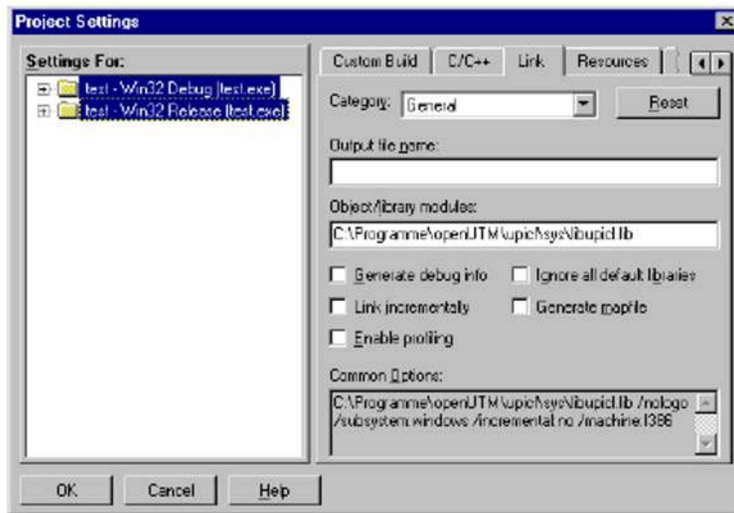
1. Link in the UPIC-local libraries `libupicl.lib` and `libxtclt.lib`:

   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.

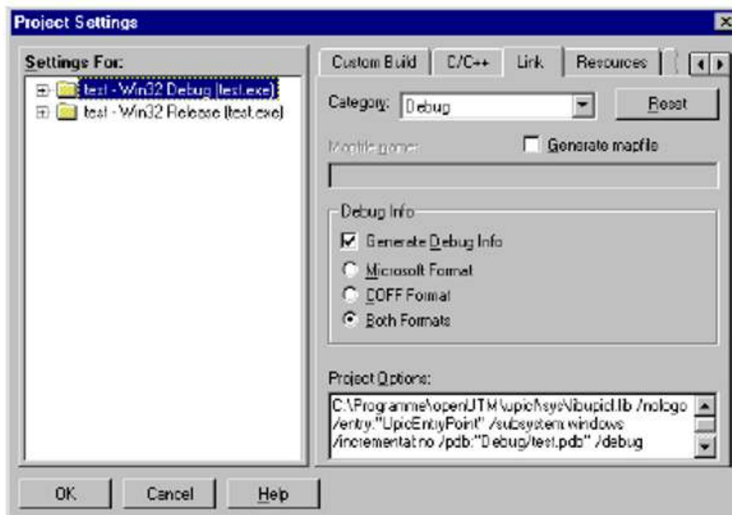
   These libraries must be entered in front of all existing `*.lib` files. *utm-dir* stands for the installation directory of openUTM. If you enter search paths in *Extras/Options* in Developer Studio, you need not type in the full pathname here.



2. Configure debugger information:

Select the *Link* tab sheet and in the *Settings For* list box mark *Win32Debug* or *x64 Debug* 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**.

6.1.2 Implementation on Unix and Linux systems

When creating and implementing CPI-C applications, take into account the special features described in section "Compilation, linking, starting on Unix and Linux systems" and section “Runtime environment, environment variables on Unix and Linux systems”.
When creating and implementing UPIC-local applications in Unix and Linux systems, you must also take into account the specifications described in section “Special features implementing UPIC local on Unix and Linux systems”.

6.1.2.1 Compilation, linking, starting on Unix and Linux systems

When compiling and linking CPI-C applications on Unix and Linux systems, you must observe the following:

- Every CPI-C program requires the following header file for compilation:

  `#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 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` which can be found in the *upic-dir*`/sys/` directory.

  For programs which do not use PCMX:
  - the system libraries `nsl.so and dl.so.` On a few systems `socket.so` also
  - the library `libupicsoc` which can be found in the *upic-dir*`/sys/` directory.

  For programs which do not use PCMX and use multi-threading:
  - the system libraries `nsl.so`, `dl.so` and `socket.so`
  - the library `libupicsocmt` 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 and Linux systems by entering the program name (note that the UTM application must be started beforehand).

6.1.2.2 Runtime environment, environment variables on Unix and Linux systems

The environment variables listed in the table on "Runtime environment, linking, starting" must be set in order to operate CPI-C applications:

You can set the environment variables as follows:

```
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 in a CPI-C program are allowed 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!

6.1.2.3 Special features when using UPIC local on Unix and Linux systems

When using UPIC-local applications on Unix and Linux systems, you must also bear in mind the special features described below.

Linking UPIC local applications in Unix and Linux systems

When a CPI-C client application is connected locally to a UTM application on a Unix or Linux system, you must link in the library `libupicipc` in the directory *utmpath*`/upicl/sys` instead of `libupiccmx`.

For XATMI client programs based on UPIC-L, the library `libxtclt` from the directory *utmpath*`/upicl/xatmi/sys` is also required.

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.

6.1.3 Using on BS2000 systems

You should take note of the special considerations listed below when using CPI-C applications on BS2000 systems.

Compilation, linking, starting on BS2000 systems

The following applies when compiling and linking CPI-C applications on BS2000 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 `$`*userid*`.SYSLIB.UTM-CLIENT.070`.
- A CPI-C program comprises a set of modules which must be linked to form a single program. The following objects are required for linking:
  - main program of the user
  - User modules
  - For programs that wish to use CMX:
    - The system libraries *$sysid*`.SYSLNK.CRTE` and `$sysid.SYSLIB.CMX.014`
    - The libraries *$userid*`.SYSLIB.UTM-CLIENT.070.WCMX` and

      `$`*userid*`.SYSLIB.UTM-CLIENT.070`
  - For programs that wish to use Sockets:
    - The system library *$sysid*`.SYSLNK.CRTE`
    - The library *$userid*`.SYSLIB.UTM-CLIENT.070`
- You start a CPI-C program on a BS2000 system in the same way as any other program using the command `START-EXECUTABLE-PROGRAM`.
  In doing so you have to specify SHARE-SCOPE=SYSTEM-MEMORY (default at start time of the task), *NONE must not be specified!

Runtime environment on BS2000 systems

Execution of CPI-C applications on BS2000 systems is controlled by the job variables. The link names of the job variables are listed in the table on "Runtime environment, linking, starting". You can set these as follows, for example:

Setting jobvariables for UPIC client

```
/SET-JV-LINK LINK-NAME=*UPICPAT,JV-NAME=UPICPATH
/MODIFY-JV JV-CONTENTS=*LINK(LINK-NAME=UPICPAT),SET-VALUE='prefix'
/SET-JV-LINK LINK-NAME=*UPICFIL,JV-NAME=UPICFILE
/MODIFY-JV JV-CONTENTS=*LINK(LINK-NAME=UPICFIL),SET-VALUE='filename'
/SET-JV-LINK LINK-NAME=*UPICLOG,JV-NAME=UPICLOG
/MODIFY-JV JV-CONTENTS=*LINK(LINK-NAME=UPICLOG),SET-VALUE='prefix'
/SET-JV-LINK LINK-NAME=*UPICTRA,JV-NAME=UPICTRACE
/MODIFY-JV JV-CONTENTS=*LINK(LINK-NAME=UPICTRA),SET-VALUE='option'
```

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”).

6.2 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”.

6.3 Behavior in the event of errors

This section describes the effects on a communication partner when a UTM 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 configured 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 configured for the client application. In this case, several clients can connect 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 connecting 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.

UPIC local (Unix, Linux and Windows systems only):

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 and Linux 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.

<reason>	Meaning
1	When sending the rest of the data, the value of data length is negative
9	The SIGTRAP signal has occurred
10	Error when establishing the connection
11	Error when receiving confirmation for connection setup
12	Message other than connection setup received
13	Error when sending data
14	Error when receiving data
15	Invalid message received
16	Error when shutting down connection

For error diagnosis see also section “Diagnostics”.

UPIC local (Unix, Linux and Windows systems only):

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 on Unix, Linux and Windows Systems” under the dump error codes.

For error diagnosis you require the dump (e.g. core dump in Unix and Linux 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).

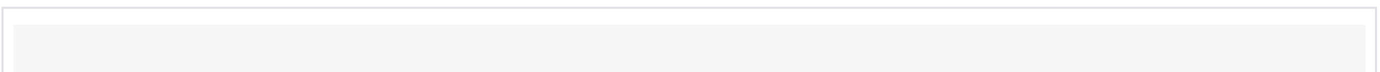
Message exchange with SYSTEM PEND ER

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”).

A separate error message for a UPIC-Client can be created 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:



On BS2000 systemen invoke the ping command as follows

```
/&* for IPV4 connections
/START-PING4 <hostname>

/&* for IPV6 connetions
/START-PING6 <hostname>
```

- Use a `ping` command to check whether it is possible at all to establish a network connection between the client and server.

  On Unix, Linux and Windows systems, call the `ping` command using:

  `ping <internetaddress>` or `ping <hostname>`

  `ping` must be in your path, i.e. the PATH variable must be suitably set.

  On BS2000 systems, call ping as follows:

  Check the TCP/IP protocol using one of the standard applications `telnet` or `ftp`.

- On Unix, Linux and Windows systems, call these commands as follows:

  `telnet` *internetaddress* or `telnet` *hostname*
  `ftp` *internetaddress* or `ftp` *hostname*

  The applications must be in your path, i.e. the PATH variable must be suitably set.

  On BS2000 systems, 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 configuration (side information if necessary) and the partner configuration (openUTM if necessary).

  BS2000 systems:

  In a configuration which requires BCMAP entries in the BS2000 system, 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.

6.4 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 PCMX trace files
- with Unix and Linux systems the core files with accompanying phases

Additional UTM documents are required for errors relating to the UTM partner application:

- KDCDEF configuration and UTM diagnostics dump of the UTM partner application
- any output logs are sent to the standard output or standard error output
- Unix, Linux and Windows systems: *stderr*, *stdout*
- BS2000 systems: SYSLST, SYSLOG, SYSOUT

6.4.1 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 `UPICL`*tid*`.UPL`, where *tid* is the thread ID. You can define which directory the log file will be stored in by means of the environment variable UPICLOG (see section “Runtime environment, environment variables on Windows systems”).

If the UPICLOG environment variable is not set, the following are interpreted in the order shown:

- 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 `%TEMP%` 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 and Linux systems

The name of the log file is `UPICL`*pid,* 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 systems

The name of the logging file is `UPICL`*tsn*, 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”).
If UPICLOG is not set, the system writes to the following logging file:

`##.`usr.tmp.`UPICL`tsn

If a UPIC process is re started on the BS2000 system without performing a LOGOFF/LOGON, the TSN *tsn* is retained. This means that the logging file is overwritten!

6.4.2 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:*

  The UPIC trace can be activated by making the appropriate setting for the UPICTRACE environment variable. If the environment variable UPICTRACE is set, the value of the environment variable is used.

  The following options are available for UPICTRACE:

  `UPICTRACE=-S[X] [-r` *wrap*`] [-d`*pathname*`]`

- *Unix and Linux systems*:

  The UPIC trace is activated when the UPICTRACE environment variable is set as follows:

  `UPICTRACE=-S[X] [-r` *wrap*`] [-d`*pathname*`]`
  `export UPICTRACE`

- *BS2000 systems*:

  The UPIC trace is activated as follows:

  `/SET-JV-LINK LINK-NAME=*UPICTRA,JV-NAME=UPICTRACE`

  `/MODIFY-JV JV[-CONTENTS]=UPICTRACE,SET-VALUE='-S[X] [-R` *wrap*`] [-D`*prefix*`]'`

  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. It is advisable always to use this option since problems that arise are frequently related to the transport interface. The switch -SX in PCMX is an extension of the switch -S. In the case of Socket communication, this switch does not provide any additional effects compared to the switch -S.
-r *wrap* (Unix, Linux and Windows systems) -R *wrap* (BS2000 systems)	The decimal number specified in wrap is multiplied by BUFSIZ. This results in the maximum size of the trace file in bytes. BUFSIZ is a value which is depending on the platform and the compiler.

Maximum value of *wrap*: 2^31 (maximum value of an int)
Default value of *wrap*: 128

-d*pathname* (Unix, Linux and Windows systems)

-D*prefix* (BS2000 systems)

The path name (or the prefix) can be specified with spaces. If spaces are used, then the path name (or prefix) must be enclosed in double quotes. Double quotes can also be used if there are no spaces 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 and Linux 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 no trace is written.

BS2000 systems:

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.UPICT`*tsn* 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:

Name of the	Windows systems	Unix and Linux systems		BS2000 systems

			Unix and Linux systems if threads are used in programs	
1st file	UPICT*tid*[1].upt	UPICT*pid*[2]	UPICT*pid*[2].*tid*[1]	UPICT*tsn*[3]
2nd file	UPICU*tid1*.upt	UPICU*pid*[2]	UPIUT*pid*[2].*tid*[1]	UPICU*tsn*[3]

[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 <-> 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* or
  - 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=
    ```
- *Unix and Linux systems:*

  ```
  UPICTRACE=
  export UPICTRACE
  ```
- *BS2000 systems*:
  - 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.

6.4.3 PCMX diagnostics (Windows systems)

PCMX diagnostics are controlled by the program `cmxtrace.exe` (32-bit) or `cmxtrc64.exe` (64-bit). You can call this program in the Windows program group PCMX-32 or PCMX-64 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.

7 Examples

This chapter contains notes on the sample programs supplied, the description of the programs `UpicAnalyzer` and `UpicReplay`, as well as some simple configuration examples for linking a CPI-C application on Windows systems with openUTM on BS2000 systems, Unix, Linux and Windows systems.

7.1 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, 32-bit only.
tpcall	Complete XATMI program
upic-cob	A Cobol project

In addition, the local definition file *tpcall.ldf.smp* is provided, from which the tool XATMIGEN creates a local configuration file for the XATMI program *tpcall*.

uptac, *utp32*, *tpcall* are ready to run after a minimum of preparation. To call them, double-click, for example, on the corresponding icons which appear in the **Fujitsu Software openUTM-Client** <variant>program window after installation.

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.

7.1.1 uptac (Windows systems)

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\uptac` after installation:

File name	Type of file
uptac.c	C source code for the program; can be printed out
uptac.vcxprojuptac.sln	Microsoft Visual C++ Developer Studio project file for creating an ".exe" file (including Solution File)
uptac.exe	Executable uptac program
uptac.bat	Batch file for uptac.exe

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` (see the model entries in the `upicfile` under *upic-dir*, which are also supplied):

Side information file:

`LN.DEFAULT UPIC0000`

`SD.DEFAULT SMP30111.`*unixhost* `PORT=30111`

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. on a BS2000 system), 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 or Linux 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.

7.1.2 utp32 (Windows systems)

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.

> i *utp32* is only available as a 32-bit variant.

7.1.3 tpcall (Windows systems)

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 subdirectory `xatmi\samples` after installation

File name	Type of file
tpcall.c	C source code for the program; can be printed out
tpcall.vcxproj	Microsoft Visual C++ Developer Studio project file for creating an “.exe” file
tpcall.exe	Executable tpcall program

Before using *tpcall* to communicate with the sample application, you must first:

- make entries in the `upicfile` as with *uptac* (see section “uptac (Windows systems)”)
- create a local configuration file by clicking on the XATMIGEN symbol in the **Fujitsu Software openUTM-Client** <variant> program window.

  The supplied local definition file `xatmi\samples\tpcall.ldf.smp` 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.ldf.smp`
(SVCU ... DEST statement, see also section “Configuring UPIC”).

7.1.4 upic-cob (Windows systems)

The directory `samples\upic-cob` contains a sample project to create a UPIC-Cobol application. The example was developed using a MicroFocus Cobol compiler.

7.2 UpicAnalyzer and UpicReplay on 64-bit Linux systems

The programs *UpicAnalyzer* and *UpicReplay* are components of the Workload Capture & Replay function. Workload Capture & Replay is a multi-component program package that is used for UTM application load simulation.

These two programs, *UpicAnalyzer* and *UpicReplay* ,are briefly described below. The concept underlying Workload Capture & Replay, as well as further details, can be found in the platform-specific openUTM manual “Using UTM Applications”.

- The *UpicAnalyzer* program must be compatible with the openUTM version which has been used for capturing. „openUTM-Client V7.0 for the UPIC carrier system“ is compatible with openUTM V7.0, for example.
- The version of the *UpicReplay* program can only process input files which have been created using the same version of the *UpicAnalyzer* program.

7.2.1 UpicAnalyzer (64-bit Linux systems)

UpicAnalyzers reads the trace records from a BTRACE trace, filters out the UPIC trace records, prepares these and writes them to a file in a specific format (UPIC ReplayFile Layout).

> **VORSICHT!**
>
> The UpicAnalyzer must not find any encrypted data or passwords, otherwise no UPIC Replay can be generated.
>
> `For this purpose, no RSA keys may be generated during the UTM generation.`

7.2.2 UpicReplay (64-bit Linux systems)

To perform the operation on a Linux system, you need the side information file upicfile containing at least one entry with the name UPREPLAY.

Examples for a upicfile entry

Replay with the TAC DEMO. The UTM application UTMTEST1 runs on the computer HOST5678.

- BS2000 systems:

  `SDUPREPLAY UTMTEST1.HOST5678 DEMO LISTENER-PORT=102 T-TSEL-Format=T`

- Unix, Linux and Windows systems:

  `DUPREPLAY UTMTEST1.HOST5678 DEMO LISTENER-PORT=11111 T-TSEL-Format=T`

UTMTEST1 must have been configured either in MAX APPLINAME or in a BCAMAPPL statement. Please refer to the correponding openUTM manual “Using UTM Applications” for details.

Calling UpicReplay

UpicReplay is called as follows from a Linux shell:

UpicReplay is invoked as follows from a Linux-Shell:

```
UpicReplay InputFileName [-c<numberOfClients>] [-s<speedPercentage>] [-d[d]]
```

InputFileName	Name of the UPIC ReplayFile that you have created with UpicAnalyzer. Mandatory parameter.
-c<numberOfClients>	*numberOfClients* specifies the number of UPIC clients for which the recorded conversations are to be replayed. Default: 1, (corresponds to *-c1*) i.e. only one client is simulated. The actual limit depends on the relevant system limit.
-s<speedPercentage>	*speedPercentage* specifies the replay speed as a percentage of the original speed. This makes it possible to simulate long and short thinking times. Default: 100 (corresponds to *-s100)* i.e. original speed. *-s200* means 200%, i.e. twice the speed, achieved by halving the thinking time.
-d	Enable debug output to *stderr*, i.e. debug messages are output on thread configuration and there are fewer messages on send and receive calls.
-dd	Enables extended debug output to *stderr*, i.e. detailed debug messages are output. This option is only intended for internal *UpicReplay* diagnostic purposes. *-dd* is only of value when simulating a small number of clients. Default: no debug output.

Example

The UPIC conversations recorded in the file *Replay.1239* are to be replayed at normal speed for 100 clients. The call is as follows:

```
UpicReplay Replay.1239 -c100
```

7.3 Configuration UPIC on Windows systems <-> openUTM on BS2000 systems

The following configuration example explains the principle of configuring a link between a CPI-C application on a Windows system and a UTM application on a BS2000 system. 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`.

7.3.1 Configuration on the Windows system

UPIC parameter

```
Enable_UTM_UPIC "UPICTTY"
Initialize_Conversation "sampladm"
```

Side information file C:\UPIC\UPICFILE:

```
* UTM(BS2000) application
SDsampladm UTMUPICR.HOST456 KDCHELP
SDsampladm UTMUPICR.HOST456 KDCHELP HOSTNAME=HOST456 T-SEL=UTMUPICR PORT=102
* or, if automatic conversion of the user data is required
HDsampladm UTMUPICR.HOST456 KDCHELP
```

7.3.2 UTM Configuration on the BS2000 system

In the example, `EXAMPLE` is the BCAM-name of the remote system hosting the UPIC client.

KDCDEF generation for the UTM application on the BS2000 system

```
BCAMAPPL UTMUPICR, T-PROT=ISO
PTERM    UPICTTY, PTYPE=UPIC-R, LTERM=UPIC,
                  BCAMAPPL=UTMUPICR, PRONAM=EXAMPLE
LTERM    UPIC, USER=UPICUSER
USER     UPICUSER, STATUS=ADMIN
```

7.4 Configuration UPIC on Windows systems <-> openUTM on Unix or Linux systems

The following configuration example explains the principle of configuring a link between a CPI-C application on a Windows system and an UTM application on a Unix or Linux system. Linking via RFC1006 is shown here.

In the example, the Windows system has the symbolic host name `HOST123`; the Unix or Linux system host has the name `HOST789`.

7.4.1 UPIC Configuration on the Windows system

UPIC-Parameter

```
Enable_UTM_UPIC "UPIC0000"
Initialize_Conversation "sampladm"
```

Side Information Datei C:\UPIC\UPICFILE

```
* UPIC client on Windows-System
LNUPIC0000 UPICTTY
* partner RFC1006
SDsampladm UTMUPICR.HOST789 KDCHELP HOSTNAME=HOST789 T-SEL=UTMUPICR PORT=1230
```

7.4.2 UTM Configuration on the Unix or Linux system

KDCDEF configuration for the UTM application on Unix or Linux system

```
BCAMAPPL UTMUPICR
PTERM    UPICTTY, PTYPE=UPIC-R, LTERM=UPIC,
                  BCAMAPPL=UTMUPICR, PRONAM=HOST123
LTERM    UPIC, USER=UPICUSER
USER     UPICUSER, STATUS=ADMIN
```

8 Appendix

This chapter contains the following information:

- differences from the X/Open CPI-C interface
- character set tables
- state tables

8.1 Differences between 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_Conversation_Encryption_Level
  Extract_Cursor_Offset
  Extract_Convertion
  Extract_Max_Partner_Index
  Extract_Partner_LU_Name_Ex
  Extract_Secondary_Return_Code
  Extract_Shutdown_State
  Extract_Shutdown_Time
  Extract_Transaction_State
  Disable_UTM_UPIC
  Set_Allocate_Timer
  Set_Client_Context
  Set_Conversation_Encryption_Level
  Set_Conversation_New_Password
  Set_Convertion
  Set_Function_Key
  Set_Partner_Host_Name
  Set_Partner_Index
  Set_Partner_IP_Address
  Set_Partner_Port
  Set_Partner_Tsel
  Set_Partner_Tsel_Format
  Set_Receive_Timer
  Specify_Local_Port
  Specify_Local_Tsel
  Specify_Local_Tsel_Format
  Specify_Secondary_Return_Code

  The *Enable_UTM_UPIC* and *Disable_UTM_UPIC* functions regulate 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 connect to a UTM application. For further details, see section “CPI-C calls in UPIC” and chapter “Configuration”.

- 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 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 or Linux system based on ASCII code and with a UTM application on a BS2000 system 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 73 characters long; for a local connection via UPIC local (Unix, Linux and Windows system) it can only be up to 8 characters.
- The name for *TP_name* can be up to 8 characters long.

The prototypes, parameter types, and constants are described in detail in the header file *upic.h*.

8.2 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.

Variable	Character set
sym_dest_name	Set 1

Character	Character set	
	Set 1	**Set 2**
.	X	X
<	X	X
(	X	X
+		X
&		X
*		X
)		X
;		X
-		X
/		X
,		X
%		X
-		X
>		X
?		X
:		X
'		X
=		X
"		X
a-z		X
A-Z		X
0-9		X

Table 16: Character sets

T.61 character set

	0	1	2	3	4	5	6	7	8	9	...	F
0			SP	0	@	P		p				
1			!	1	A	Q	a	q				
2			"	2	B	R	b	r				
3			#	3	C	S	c	s				
4			¤	4	D	T	d	t				
5			%	5	E	U	e	u				
6			&	6	F	V	f	v				
7				7	G	W	g	w				
8	BS		(	8	H	X	h	x				
9		SS2	)	9	I	Y	i	y				
A	LF	SUB	*	:	J	Z	j	z				
B		ESC	+	;	K	[	k		PLD	CSI		
C	FF		,	<	L		l	|	PLU			
D	CR	SS3	-	=	M	]	m					
E	LS1		.	>	N		n					
F	LS0		/	?	O	-	o					

Table 17: Code table T.61 in accordance with CCITT recommodation

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

8.3 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.

Call	Result	Follow-up state, if previously in state				
		Start	Reset	Init.	Send	Receive
Initialize_Conversation	ok	psc	Init.	psc	psc	psc
Initialize_Conversation	pc	psc	-	psc	psc	psc
Initialize_Conversation	ps	psc	-	psc	psc	psc
Allocate	ok	psc	psc	Send	psc	psc
Allocate	ae	psc	psc	Reset	psc	psc
Allocate	pc	psc	psc	-	psc	psc
Allocate	pe	psc	psc	-	psc	psc
Allocate	ps	psc	psc	-	psc	psc
Deallocate	ok	psc	psc	Reset	Reset	Reset
Deallocate	pc	psc	psc	-	-	-
Deallocate	ps	psc	psc	-	-	-
Deferred_Deallocate	-	-	-	-	-	-
Extract_Client_Context	ok	psc	-	-	-	-
Extract_Client_Context	pc	psc	-	-	-	-
Extract_Client_Context	ps	psc	-	-	-	-
Extract_Conversation_Encryption_Level	ok	psc	psc	-	-	-
Extract_Conversation_Encryption_Level	pc	psc	psc	-	-	-
Extract_Conversation_Encryption_Level	ps	psc	psc	-	-	-
Extract_Conversation_State	ok	psc	psc	-	-	-
Extract_Conversation_State	pc	psc	psc	-	-	-
Extract_Conversation_State	ps	psc	psc	-	-	-
Extract_Convertion	ok	psc	psc	-	psc	psc
Extract_Convertion	pc	psc	psc	-	psc	psc
Extract_Convertion	ps	psc	psc	-	psc	psc

Extract_Cursor_Offset	ok	psc	-[1]	-	-	-
Extract_Cursor_Offset	pc	psc	-	-	-	-
Extract_Cursor_Offset	ps	psc	-	-	-	-
Extract_Max_Partner_Index	ok	-	-	-	-	-
Extract_Max_Partner_Index	pc	-	-	-	-	-
Extract_Max_Partner_Index	ps	-	-	-	-	-
Extract_Partner_LU_Name	ok	-	-	-	-	-
Extract_Partner_LU_Name	pc	-	-	-	-	-
Extract_Partner_LU_Name	ps	-	-	-	-	-
Extract_Partner_LU_Name_Ex	ok	-	-	-	-	-
Extract_Partner_LU_Name_Ex	pc	-	-	-	-	-
Extract_Partner_LU_Name_Ex	ps	-	-	-	-	-
Extract_Secondary_Information	ok	-	-	-	-	-
Extract_Secondary_Information	pc	-	-	-	-	-
Extract_Secondary_Information	ps	-	-	-	-	-
Extract_Secondary_Return_Code	ok	psc	psc	-	-	-
Extract_Secondary_Return_Code	nr	psc	psc	-	-	-
Extract_Secondary_Return_Code	pc	psc	psc	-	-	-
Extract_Secondary_Return_Code	ps	psc	psc	-	-	-
Extract_Shutdown_State	ok	psc	-[1]	psc	-	-
Extract_Shutdown_State	pc	psc	-[1]	psc	-	-
Extract_Shutdown_State	ps	psc	-[1]	psc	-	-
Extract_Shutdown_Time	ok	psc	-[1]	psc	-	-
Extract_Shutdown_Time	pc	psc	-[1]	psc	-	-
Extract_Shutdown_Time	ps	psc	-[1]	psc	-	-
Extract_Transaction_State	ok	psc	-[1]	psc	-	-
Extract_Transaction_State	pc	psc	-[1]	psc	-	-
Extract_Transaction_State	ps	psc	-[1]	psc	-	-
Prepare_To_Receive	ok	psc	psc	psc	Receive	-

Prepare_To_Receive	da	psc	psc	psc	Reset	psc
Prepare_To_Receive	pc	psc	psc	psc	-	psc
Prepare_To_Receive	rf	psc	psc	psc	Reset	psc
Receive / Receive_Mapped_Data	ok{dr,no}	psc	psc	psc	Receive	-
Receive / Receive_Mapped_Data	ok{nd,se}	psc	psc	psc	-	Send
Receive / Receive_Mapped_Data	ok{dr,se}	psc	psc	psc	-	Send
Receive / Receive_Mapped_Data	ae	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	da	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	dn	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	rf	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	oi,un	psc	psc	psc	Receive	-
Receive / Receive_Mapped_Data	pc	psc	psc	psc	-	-
Receive / Receive_Mapped_Data	ps	psc	psc	psc	-	-
Send_Data / Send_Mapped_Data	ok	psc	psc	psc	-	psc
Send_Data / Send_Mapped_Data	ae	psc	psc	psc	Reset	psc
Send_Data / Send_Mapped_Data	da	psc	psc	psc	Reset	psc
Send_Data / Send_Mapped_Data	pc	psc	psc	psc	-	psc
Send_Data / Send_Mapped_Data	rf	psc	psc	psc	Reset	psc
Set_Allocate_Timer	ok	psc	psc	-	psc	psc
Set_Allocate_Timer	pc	psc	psc	-	psc	psc
Set_Allocate_Timer	ps	psc	psc	-	psc	psc
Set_Client_Context	ok	psc	psc	psc	-	psc
Set_Client_Context	pc	psc	psc	psc	-	psc
Set_Client_Context	ps	psc	psc	psc	-	psc
Set_Conversation_Encryption_Level	ok	psc	psc	-	psc	psc
Set_Conversation_Encryption_Level	pc	psc	psc	-	psc	psc
Set_Conversation_Encryption_Level	ps	psc	psc	-	psc	psc
Set_Convertion	ok	psc	psc	-	psc	psc
Set_Convertion	pc	psc	psc	-	psc	psc

Set_Convertion	ps	psc	psc	-	psc	psc
Set_Conversation_Security_Type	ok	psc	psc	-	psc	psc
Set_Conversation_Security_Type	pc	psc	psc	-	psc	psc
Set_Conversation_Security_Type	pn	psc	psc	-	psc	psc
Set_Conversation_Security_New_Password	ok	psc	psc	-	psc	psc
Set_Conversation_Security_New_Password	pc	psc	psc	-	psc	psc
Set_Conversation_Security_Password	ok	psc	psc	-	psc	psc
Set_Conversation_Security_Password	pc	psc	psc	-	psc	psc
Set_Conversation_Security_User_ID	ok	psc	psc	-	psc	psc
Set_Conversation_Security_User_ID	pc	psc	psc	-	psc	psc
Set_Deallocate_Type	ok	psc	psc	-	-	-
Set_Deallocate_Type	pc	psc	psc	-	-	-
Set_Deallocate_Type	ps	psc	psc	-	-	-
Set_Function_Key	ok	psc	psc	psc	-	-
Set_Function_Key	pc	psc	psc	psc	-	-
Set_Function_Key	ps	psc	psc	psc	-	-
Set_Receive_Timer	ok	psc	psc	psc	-	-
Set_Receive_Timer	pc	psc	psc	psc	-	-
Set_Receive_Timer	ps	psc	psc	psc	-	-
Set_Receive_Type	ok	-	-	-	-	-
Set_Receive_Type	pc	-	-	-	-	-
Set_Partner_Host_Name	ok	psc	psc	-	psc	psc
Set_Partner_Host_Name	pc	psc	psc	-	psc	psc
Set_Partner_Host_Name	ps	psc	psc	-	psc	psc
Set_Partner_Index	ok	psc	psc	-	psc	psc
Set_Partner_Index	pc	psc	psc	-	psc	psc
Set_Partner_Index	ps	psc	psc	-	psc	psc
Set_Partner_IP_Address	ok	psc	psc	-	psc	psc
Set_Partner_IP_Address	pc	psc	psc	-	psc	psc

Set_Partner_IP_Address	ps	psc	psc	-	psc	psc
Set_Partner_LU_Name	ok	psc	psc	-	psc	psc
Set_Partner_LU_Name	pc	psc	psc	-	psc	psc
Set_Partner_LU_Name	ps	psc	psc	-	psc	psc
Set_Partner_Port	ok	psc	psc	-	psc	psc
Set_Partner_Port	pc	psc	psc	-	psc	psc
Set_Partner_Port	ps	psc	psc	-	psc	psc
Set_Partner_Tsel	ok	psc	psc	-	psc	psc
Set_Partner_Tsel	pc	psc	psc	-	psc	psc
Set_Partner_Tsel	ps	psc	psc	-	psc	psc
Set_Partner_Tsel_Format	ok	psc	psc	-	psc	psc
Set_Partner_Tsel_Format	pc	psc	psc	-	psc	psc
Set_Partner_Tsel_Format	ps	psc	psc	-	psc	psc
Set_Sync_Level	ok	psc	-	psc	psc	psc
Set_Sync_Level	pc	psc	-	psc	psc	psc
Set_Sync_Level	ps	psc	-	psc	psc	psc
Set_TP_Name	ok	psc	psc	-	psc	psc
Set_TP_Name	pc	psc	psc	-	psc	psc
Specify_Local_Port	ok	psc	-	psc	psc	psc
Specify_Local_Port	pc	psc	-	psc	psc	psc
Specify_Local_Port	ps	psc	-	psc	psc	psc
Specify_Local_Tsel	ok	psc	-	psc	psc	psc
Specify_Local_Tsel	pc	psc	-	psc	psc	psc
Specify_Local_Tsel	ps	psc	-	psc	psc	psc
Specify_Local_Tsel_Format	ok	psc	-	psc	psc	psc
Specify_Local_Tsel_Format	pc	psc	-	psc	psc	psc
Specify_Local_Tsel_Format	ps	psc	-	psc	psc	psc
Specify_Secondary_Return_Code	ok	psc	-	-	-	-
Specify_Secondary_Return_Code	pc	psc	-	-	-	-

Specify_Secondary_Return_Code	ps	psc	-	-	-	-
Enable_UTM_UPIC	ok	Reset	psc	psc	psc	psc
Enable_UTM_UPIC	pc	-	psc	psc	psc	psc
Enable_UTM_UPIC	ps	-	psc	psc	psc	psc
Disable_UTM_UPIC	ok	psc	Start	Start	Start	Start
Disable_UTM_UPIC	pc	psc	-	-	-	-
Disable_UTM_UPIC	ps	psc	-	-	-	-

Table 19: State table for CPI-C calls

[1]Permitted only directly after a *Receive()* / *Receive_Mapped_Data()* call

Abbreviations for the state table:

Result	Return codes
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)

Result	data_received and status_received:
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)

Follow-up state	Meaning
-	No state change
psc	Error CM_PROGRAM_STATE_CHECK

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.

9 Glossary

A term in *italic* font means that it is explained somewhere else in the glossary.

abnormal termination of a UTM application

Termination of a *UTM application*, where the *KDCFILE* is not updated. Abnormal termination is caused by a serious error, such as a crashed computer or an error in the system software. If you then restart the application, openUTM carries out a *warm start*.

abstract syntax (OSI)

Abstract syntax is defined as the set of formally described data types which can be exchanged between applications via *OSI TP*. Abstract syntax is independent of the hardware and programming language used.

acceptor (CPI-C)

The communication partners in a *conversation* are referred to as the *initiator* and the acceptor. The acceptor accepts the conversation initiated by the initiator with Accept_Conversation.

access list

An access list defines the authorization for access to a particular *service*, *TAC queue* or *USER queue*. An access list is defined as a *key set* and contains one or more *key codes*, each of which represent a role in the application. Users or LTERMs or (OSI) LPAPs can only access the service or *TAC queue*/*USER queue* when the corresponding roles have been assigned to them (i.e. when their *key set* and the access list contain at least one common *key code)*.

access point (OSI)

See *service access point.*

ACID properties

Acronym for the fundamental properties of *transactions*: atomicity, consistency, isolation and durability.

administration

Administration and control of a *UTM application* by an *administrator* or an *administration program*.

administration command

Commands used by the *administrator* of a *UTM application* to carry out administration functions for this application. The administration commands are implemented in the form of *transaction codes*.

administration journal

See *cluster administration journal*.

administration program

Program unit containing calls to the *program interface for administration*. This can be either the standard administration program *KDCADM* that is supplied with openUTM or a program written by the user.

administrator

User who possesses administration authorization.

AES

AES (Advanced Encryption Standard) is the current symmetric encryption standarddefined by the National Institute of Standards and Technology (NIST) and based on the Rijndael algorithm developed at the University of Leuven (Belgium). If the AES method is used, the UPIC client generates an AES key for each session.

Apache Axis

Apache Axis (Apache eXtensible Interaction System) is a SOAP engine for the design of Web services and client applications. There are implementations in C++ and Java.

Apache Tomcat

Apache Tomcat provides an environment for the execution of Java code on Web servers. It was developed as part of the Apache Software Foundation's Jakarta project. It consists of a servlet container written in Java which can use the JSP Jasper compiler to convert JavaServer pages into servlets and run them. It also provides a fully featured HTTP server.

application cold start

See *cold start*.

application context (OSI)

The application context is the set of rules designed to govern communication between two applications. This includes, for instance, abstract syntaxes and any assigned transfer syntaxes.

application entity (OSI)

An application entity (AE) represents all the aspects of a real application which are relevant to communications. An application entity is identified by a globally unique name (“globally” is used here in its literal sense, i.e. worldwide), the *application entity title* (AET). Every application entity represents precisely one *application process*. One application process can encompass several application entities.

application entity qualifier (OSI)

Component of the *application entity title*. The application entity qualifier identifies a *service access point* within an application. The structure of an application entity qualifier can vary. openUTM supports the type “number”.

application entity title (OSI)

An application entity title is a globally unique name for an *application entity* (“globally” is used here in its literal sense, i.e. worldwide). It is made up of the *application process title* of the relevant *application process* and the *application entity qualifier.*

application information

This is the entire set of data used by the *UTM application*. The information comprises memory areas and messages of the UTM application including the data currently shown on the screen. If operation of the UTM application is coordinated with a database system, the data stored in the database also forms part of the application information.

application process (OSI)

The application process represents an application in the *OSI reference model*. It is uniquely identified globally by the *application process title*.

application process title (OSI)

According to the OSI standard, the application process title (APT) is used for the unique identification of applications on a global (i.e. worldwide) basis. The structure of an application process title can vary. openUTM supports the type *Object Identifier*.

application program

An application program is the core component of a *UTM application*. It comprises the main routine *KDCROOT* and any *program units* and processes all jobs sent to a *UTM application*.

application restart

see *warm start*

application service element (OSI)

An application service element (ASE) represents a functional group of the application layer (layer 7) of the *OSI reference model*.

application warm start

see *warm start*.

association (OSI)

An association is a communication relationship between two application entities. The term “association” corresponds to the term *session* in *LU6.1*.

asynchronous conversation

CPI-C conversation where only the *initiator* is permitted to send. An asynchronous transaction code for the *acceptor* must have been generated in the *UTM application*.

asynchronous job

Job carried out by the job submitter at a later time. openUTM includes *message queuing* functions for processing asynchronous jobs (see *UTM-controlled queue* and *service-controlled queue)*. An asynchronous job is described by the *asynchronous message*, the recipient and, where applicable, the required execution time. If the recipient is a terminal, a printer or a transport system application, the asynchronous job is a *queued output job*. If the recipient is an *asynchronous service* of the same application or a remote application, the job is a *background job*. Asynchronous jobs can be *time-driven jobs* or can be integrated in a *job complex*.

asynchronous message

Asynchronous messages are messages directed to a *message queue*. They are stored temporarily by the local *UTM application* and then further processed regardless of the job submitter. Distinctions are drawn between the following types of asynchronous messages, depending on the recipient:

- In the case of asynchronous messages to a *UTM-controlled queue,* all further processing is controlled by openUTM. This type includes messages that start a local or remote *asynchronous service* (see also *background job*) and messages sent for output on a terminal, a printer or a transport system application (see also *queued output job*).
- In the case of asynchronous messages to a *service-controlled queue*, further processing is controlled by a *service* of the application. This type includes messages to a *TAC queue*, messages to a *USER queue* and messages to a *temporary queue*. The USER queue and the temporary queue must belong to the local application, whereas the TAC queue can be in both the local application and the remote application.

asynchronous program

Program unit started by a *background job*.

asynchronous service (KDCS)

Service which processes a *background job*. Processing is carried out independently of the job submitter. An asynchronous service can comprise one or more program units/transactions. It is started via an asynchronous *transaction code*.

audit (BS2000 systems)

During execution of a *UTM application,* UTM events which are of relevance in terms of security can be logged by *SAT* for auditing purposes.

authentication

See *system access control*.

authorization

See *data access control*.

Axis

See *Apache Axis*.

background job

Background jobs are *asynchronous jobs* destined for an *asynchronous service* of the current application or of a remote application. Background jobs are particularly suitable for time-intensive processing or processing which is not time-critical and where the results do not directly influence the current dialog.

basic format

Format in which terminal users can make all entries required to start a service.

basic job

Asynchronous job in a *job complex*.

browsing asynchronous messages

A *service* sequentially reads the *asynchronous messages* in a *service-controlled queue*. The messages are not locked while they are being read and they remain in the queue after they have been read. This means that they can be read simultaneously by different services.

bypass mode (BS2000 systems)

Operating mode of a printer connected locally to a terminal. In bypass mode, any *asynchronous message* sent to the printer is sent to the terminal and then redirected to the printer by the terminal without being displayed on screen.

cache

Used for buffering application data for all the processes of a *UTM application*.
The cache is used to optimize access to the *page pool* and, in the case of UTM cluster applications, the *cluster page pool*.

CCR (Commitment, Concurrency and Recovery)

CCR is an Application Service Element (ASE) defined by OSI used for OSI TP communication which contains the protocol elements (services) related to the beginning and end (commit or rollback) of a *transaction*. CCR supports the two-phase commitment.

CCS name (BS2000 systems)

See *coded character set name*.

client

Clients of a *UTM application* can be:

- terminals
- UPIC client programs
- transport system applications (e.g. DCAM, PDN, CMX, socket applications or UTM applications which have been generated as *transport system applications*).

Clients are connected to the UTM application via LTERM partners.
Note: UTM clients which use the OpenCPIC carrier system are treated just like *OSI TP partners*.

client side of a conversation

This term has been superseded by *initiator*.

cluster

A number of computers connected over a fast network and which in many cases can be seen as a single computer externally. The objective of clustering is generally to increase the computing capacity or availability in comparison with a single computer.

cluster administration journal

The cluster administration journal consists of:

- two log files with the extensions JRN1 and JRN2 for global administration actions,
- the JKAA file which contains a copy of the KDCS Application Area (KAA). Administrative changes that are no longer present in the two log files are taken over from this copy.

The administration journal files serve to pass on to the other node applications those administrative actions that are to apply throughout the cluster to all node applications in a UTM cluster application.

cluster configuration file

File containing the central configuration data of a *UTM cluster application*. The cluster configuration file is created using the UTM generation tool *KDCDEF*.

cluster filebase

Filename prefix or directory name for the *UTM cluster files*.

cluster GSSB file

File used to administer GSSBs in a *UTM cluster application*. The cluster GSSB file is created using the UTM generation tool *KDCDEF*.

cluster lock file

File in a *UTM cluster application* used to manage cross-node locks of user data areas.

cluster page pool

The cluster page pool consists of an administration file and up to 10 files containing a *UTM cluster application's* user data that is available globally in the cluster (service data including LSSB, GSSB and ULS). The cluster page pool is created using the UTM generation tool *KDCDEF*.

cluster start serialization file

Lock file used to serialize the start-up of individual node applications (only on Unix, Linux and Windows systems).

cluster ULS file

File used to administer the ULS areas of a *UTM cluster application*. The cluster ULS file is created using the UTM generation tool *KDCDEF*.

cluster user file

File containing the user management data of a *UTM cluster application*. The cluster user file is created using the UTM generation tool *KDCDEF*.

coded character set name (BS2000 systems)

If the product *XHCS* (e**X**tended **H**ost **C**ode **S**upport) is used, each character set used is uniquely identified by a coded character set name (abbreviation: “CCS name” or “CCSN”).

cold start

Start of a *UTM application* after the application terminates normally (*normal termination*) or after a new generation (see also *warm start*).

communication area (KDCS)

KDCS *primary storage area*, secured by transaction logging and which contains service-specific data. The communication area comprises 3 parts:

- the KB header with general service data
- the KB return area for returning values to KDCS calls
- the KB program area for exchanging data between UTM program units within a single *service*.

communication end point

see *transport system end point*

communication resource manager

In distributed systems, communication resource managers (CRMs) control communication between the application programs. openUTM provides CRMs for the international OSI TP standard, for the LU6.1 industry standard and for the proprietary openUTM protocol UPIC.

configuration

Sum of all the properties of a *UTM application*. The configuration describes:

- application parameters and operating parameters
- the objects of an application and the properties of these objects. Objects can be *program units* and *transaction codes*, communication partners, printers, *user IDs*, etc.
- defined measures for controlling data and system access.

The configuration of a UTM application is defined at generation time (*static configuration*) and can be changed dynamically by the administrator (while the application is running, *dynamic configuration*). The configuration is stored in the *KDCFILE*.

confirmation job

Component of a *job complex* where the confirmation job is assigned to the *basic job*. There are positive and negative confirmation jobs. If the *basic job* returns a positive result, the positive confirmation job is activated, otherwise, the negative confirmation job is activated.

connection bundle

see *LTERM bundle.*

connection user ID

User ID under which a *TS application* or a *UPIC client* is signed on at the *UTM application* directly after the connection has been established. The following applies, depending on the client (= LTERM partner) generation:

- The connection user ID is the same as the USER in the LTERM statement (explicit connection user ID). An explicit connection user ID must be generated with a USER statement and cannot be used as a “genuine” *user ID*.
- The connection user ID is the same as the LTERM partner (implicit connection user ID) if no USER was specified in the LTERM statement or if an LTERM pool has been generated.

In a *UTM cluster application*, the service belonging to a connection user ID (RESTART=YES in LTERM or USER) is bound to the connection and is therefore local to the node.
A connection user ID generated with RESTART=YES can have a separate service in each *node application*.

contention loser

Every connection between two partners is managed by one of the partners. The partner that manages the connection is known as the *contention winner*. The other partner is the contention loser.

contention winner

A connection's contention winner is responsible for managing the connection. Jobs can be started by the contention winner or by the *contention loser*. If a conflict occurs, i.e. if both partners in the communication want to start a job at the same time, then the job stemming from the contention winner uses the connection.

conversation

In CPI-C, communication between two CPI-C application programs is referred to as a conversation. The communication partners in a conversation are referred to as the *initiator* and the *acceptor*.

conversation ID

CPI-C assigns a local conversation ID to each *conversation*, i.e. the *initiator* and *acceptor* each have their own conversation ID. The conversation ID uniquely assigns each CPI-C call in a program to a conversation.

CPI-C

CPI-C (**C**ommon **P**rogramming **I**nterface for **C**ommunication) is a program interface for program-to-program communication in open networks standardized by X/Open and CIW (**C**PI-C **I**mplementor's **W**orkshop).
The CPI-C implemented in openUTM complies with X/Open’s CPI-C V2.0 CAE Specification. The interface is available in COBOL and C. In openUTM, CPI-C can communicate via the OSI TP, LU6.1 and UPIC protocols and with openUTM-LU62.

Cross Coupled System / XCS

Cluster of BS2000 computers with the *Highly Integrated System Complex* Multiple System Control Facility (HIPLEX® MSCF).

data access control

In data access control openUTM checks whether the communication partner is authorized to access a particular object belonging to the application. The access rights are defined as part of the configuration.

data space (BS2000 systems)

Virtual address space of BS2000 which can be employed in its entirety by the user. Only data and programs stored as data can be addressed in a data space; no program code can be executed.

dead letter queue

The dead letter queue is a TAC queue which has the fixed name KDCDLETQ.
It is always available to save queued messages sent to transaction codes, TAC queues, LPAP or OSI-LPAP partners but which could not be processed. The saving of queued messages in the dead letter queue can be activated or deactivated for each message destination individually using the TAC, LPAP or OSI-LPAP statement's DEAD-LETTER-Q parameter.

DES

DES (Data Encryption Standard) is an international standard for encrypting data. One key is used in this method for encoding and decoding. If the DES method is used, the UPIC client generates a DES key for each session.

dialog conversation

CPI-C conversation in which both the *initiator* and the *acceptor* are permitted to send. A dialog transaction code for the *acceptor* must have been generated in the *UTM application*.

dialog job, interactive job

Job which starts a *dialog service*. The job can be issued by a *client* or, when two servers communicate with each other (*server-server communication*), by a different application.

dialog message

A message which requires a response or which is itself a response to a request. The request and the response both take place within a single service. The request and reply together form a dialog step.

dialog program

Program unit which partially or completely processes a *dialog step*.

dialog service

Service which processes a *job* interactively (synchronously) in conjunction with the job submitter (*client* or another server application) . A dialog service processes *dialog messages* received from the job submitter and generates dialog messages to be sent to the job submitter. A dialog service comprises at least one *transaction.* In general, a dialog service encompasses at least one dialog step. Exception: in the event of *service chaining,* it is possible for more than one service to comprise a dialog step.

dialog step

A dialog step starts when a *dialog message* is received by the *UTM application*. It ends when the UTM application responds.

dialog terminal process (Unix , Linux and Windows systems)

A dialog terminal process connects a terminal of a Unix, Linux or Windows system with the work processes of the *UTM application*. Dialog terminal processes are started either when the user enters utmdtp or via the LOGIN shell. A separate dialog terminal process is required for each terminal to be connected to a UTM application.

distributed processing

Processing of *dialog jobs* by several different applications or the transfer of *background jobs* to another application. The higher-level protocols *LU6.1* and *OSI TP* are used for distributed processing. openUTM-LU62 also permits distributed processing with LU6.2 partners. A distinction is made between distributed processing with *distributed transactions* (transaction logging across different applications) and distributed processing without distributed transactions (local transaction logging only). Distributed processing is also known as server-server communication.

distributed transaction

Transaction which encompasses more than one application and is executed in several different (sub-)transactions in distributed systems.

distributed transaction processing

Distributed processing with *distributed transactions.*

dynamic configuration

Changes to the *configuration* made by the administrator. UTM objects such as *program units*, *transaction codes*, *clients*, *LU6.1 connections,* printers or *user IDs* can be added, modified or in some cases deleted from the configuration while the application is running. To do this, it is necessary to create separate *administration programs* which use the functions of the *program interface for administration*. The WinAdmin administration program or the WebAdmin administration program can be used to do this, or separate *administration programs* must be created that utilize the functions of the *administration program interface.*

encryption level

The encryption level specifies if and to what extent a client message and password are to be encrypted.

event-driven service

This term has been superseded by *event service*.

event exit

Routine in an application program which is started automatically whenever certain events occur (e.g. when a process is started, when a service is terminated). Unlike *event services*, an event exit must not contain any KDCS, CPI-C or XATMI calls.

event function

Collective term for *event exits* and *event services*.

event service

Service started when certain events occur, e.g. when certain UTM messages are issued. The *program units* for event-driven services must contain KDCS calls.

filebase

UTM application filebase
On BS2000 systems, filebase is the prefix for the *KDCFILE*, the *user log file* USLOG and the *system log file* SYSLOG.
On Unix, Linux and Windows systems, filebase is the name of the directory under which the KDCFILE, the user log file USLOG, the system log file SYSLOG and other files relating to the UTM application are stored.

Functional Unit (FU)

A subset of the *OSI TP* protocol providing a particular functionality. The OSI TP protocol is divided into the following functional units:

- Dialog
- Shared Control
- Polarized Control
- Handshake
- Commit
- Chained Transactions
- Unchained Transactions
- Recovery

Manufacturers implementing OSI TP need not include all functional units, but can concentrate on a subset instead. Communications between applications of two different OSI TP implementations is only possible if the included functional units are compatible with each other.

generation

See *UTM generation.*

global secondary storage area

See *secondary storage area.*

hardcopy mode

Operating mode of a printer connected locally to a terminal. Any message which is displayed on screen will also be sent to the printer.

heterogeneous link

In the case of *server-server communication:* a link between a *UTM application* and a non-UTM application, e.g. a CICS or TUXEDO application.

Highly Integrated System Complex / HIPLEX ®

Product family for implementing an operating, load sharing and availability cluster made up of a number of BS2000 servers.

HIPLEX ® MSCF

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)
Provides the infrastructure and basic functions for distributed applications with HIPLEX®.

homogeneous link

In the case of *server-server communication*: a link between two *UTM applications.* It is of no significance whether the applications are running on the same operating system platforms or on different platforms.

inbound conversation (CPI-C)

See *incoming conversation.*

incoming conversation (CPI-C)

A conversation in which the local CPI-C program is the *acceptor* is referred to as an incoming conversation. In the X/Open specification, the term “inbound conversation” is used synonymously with “incoming conversation”.

initial KDCFILE

In a *UTM cluster application*, this is the *KDCFILE* generated by *KDCDEF* and which must be copied for each node application before the node applications are started.

initiator (CPI-C)

The communication partners in a *conversation* are referred to as the initiator and the *acceptor*. The initiator sets up the conversation with the CPI-C calls Initialize_Conversation and Allocate.

insert

Field in a message text in which openUTM enters current values.

inverse KDCDEF

A function which uses the dynamically adapted configuration data in the *KDCFILE* to generate control statements for a *KDCDEF* run. An inverse KDCDEF can be started “offline” under *KDCDEF* or “online” via the *program interface for administration*.

IUTMDB

Interface used for the coordinated interaction with resource managers on BS2000 systems. This includes data repositories (LEASY) and data base systems (SESAM/SQL, UDS/SQL).

JConnect client

Designation for clients based on the product openUTM-JConnect. The communication with the UTM application is carried out via the *UPIC protocol* .

JDK

Java Development Kit
Standard development environment from Oracle Corporation for the development of Java applications.

job

Request for a *service* provided by a *UTM application*. The request is issued by specifying a transaction code. See also: *queued output job*, *dialog job*, *background job*, *job complex*.

job complex

Job complexes are used to assign *confirmation jobs* to *asynchronous jobs*. An asynchronous job within a job complex is referred to as a *basic job*.

job-receiving service (KDCS)

A job-receiving service is a *service* started by a *job-submitting service* of another server application.

job-submitting service (KDCS)

A job-submitting service is a *service* which requests another service from a different server application (*job-receiving service*) in order to process a job.

KDCADM

Standard administration program supplied with openUTM. KDCADM provides administration functions which are called with transaction codes (*administration commands*).

KDCDEF

UTM tool for the *generation* of *UTM applications*. KDCDEF uses the configuration information in the KDCDEF control statements to create the UTM objects *KDC FILE* and the ROOT table sources for the main routine *KDCROOT*.
In UTM cluster applications, KDCDEF also creates the *cluster configuration file,* the *cluster user file*, the *cluster page pool*, the *cluster GSSB file* and the *cluster ULS file*.

KDCFILE

One or more files containing data required for a *UTM application* to run. The KDCFILE is created with the UTM generation tool *KDCDEF*. Among other things, it contains the *configuration* of the application.

KDCROOT

Main routine of an *application program* which forms the link between the *program units* and the UTM system code. KDCROOT is linked with the *program units* to form the *application program*.

KDCS message area

For KDCS calls: buffer area in which messages or data for openUTM or for the *program unit* are made available.

KDCS parameter area

See *parameter area.*

KDCS program interface

Universal UTM program interface compliant with the national DIN 66 265 standard and which includes some extensions. KDCS (compatible data communications interface) allows dialog services to be created, for instance, and permits the use of *message queuing* functions. In addition, KDCS provides calls for *distributed processing*.

Kerberos

Kerberos is a standardized network authentication protocol (RFC1510) based on encryption procedures in which no passwords are sent to the network in clear text.

Kerberos principal

Owner of a key.
Kerberos uses symmetrical encryption, i.e. all the keys are present at two locations, namely with the key owner (principal) and the KDC (Key Distribution Center).

key code

Code that represents specific access authorization or a specific role. Several key codes are grouped into a *key set* .

key set

Group of one or more *key codes* under a particular a name. A key set defines authorization within the framework of the authorization concept used (lock/key code concept or *access list* concept). A key set can be assigned to a *user ID* , an *LTERM partner* an *(OSI) LPAP partner* , a *service* or a *TAC queue*.

linkage program

See *KDCROOT*.

local secondary storage area

See *secondary storage area.*

Log4j

Log4j is part of the Apache Jakarta project. Log4j provides information for logging information (runtime information, trace records, etc.) and configuring the log output. *WS4UTM* uses the software product Log4j for trace and logging functionality.

lock code

Code protecting an LTERM partner or transaction code against unauthorized access. Access is only possible if the *key set* of the accesser contains the appropriate *key code* (lock/key code concept).

logging process

Process in Unix, Linux and Windows systems that controls the logging of account records or monitoring data.

LPAP bundle

LPAP bundles allow messages to be distributed to LPAP partners across several partner applications. If a UTM application has to exchange a very large number of messages with a partner application then load distribution may be improved by starting multiple instances of the partner application and distributing the messages across the individual instances. In an LPAP bundle, openUTM is responsible for distributing the messages to the partner application instances. An LPAP bundle consists of a master LPAP and multiple slave LPAPs. The slave LPAPs are assigned to the master LPAP on UTM generation. LPAP bundles exist for both the OSI TP protocol and the LU6.1 protocol.

LPAP partner

In the case of *distributed processing* via the *LU6.1* protocol, an LPAP partner for each partner application must be configured in the local application. The LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned LPAP partner and not by the application name or address.

LTERM bundle

An LTERM bundle (connection bundle) consists of a master LTERM and multiple slave LTERMs. An LTERM bundle (connection bundle) allows you to distribute queued messages to a logical partner application evenly across multiple parallel connections.

LTERM group

An LTERM group consists of one or more alias LTERMs, the group LTERMs and a primary LTERM. In an LTERM group, you assign multiple LTERMs to a connection.

LTERM partner

LTERM partners must be configured in the application if you want to connect clients or printers to a *UTM application*. A client or printer can only be connected if an LTERM partner with the appropriate properties is assigned to it. This assignment is generally made in the *configuration*, but can also be made dynamically using terminal pools.

LTERM pool

The TPOOL statement allows you to define a pool of LTERM partners instead of issuing one LTERM and one PTERM statement for each *client*. If a client establishes a connection via an LTERM pool, an LTERM partner is assigned to it dynamically from the pool.

LU6.1

Device-independent data exchange protocol (industrial standard) for transaction-oriented *server-server communication*.

LU6.1-LPAP bundle

LPAP bundle for *LU6.1* partner applications.

LU6.1 partner

Partner of the *UTM application* that communicates with the UTM application via the *LU6.1* protocol. Examples of this type of partner are:

- a UTM application that communicates via LU6.1
- an application in the IBM environment (e.g. CICS, IMS or TXSeries) that communicates via LU6.1

main process (Unix /Linux / Windows systems)

Process which starts the *UTM application*. It starts the *work processes*, the *UTM system processes*, *printer processes, network processes, logging process* and the *timer process* and monitors the *UTM application*.

main routine KDCROOT

See *KDCROOT*.

management unit

SE Servers component; in combination with the *SE Manager*, permits centralized, web-based management of all the units of an SE server.

message definition file

The message definition file is supplied with openUTM and, by default, contains the UTM message texts in German and English together with the definitions of the message properties. Users can take this file as a basis for their own message modules.

message destination

Output medium for a *message*. Possible message destinations for a message from the openUTM transaction monitor include, for instance, terminals, *TS applications*, the *event service* MSGTAC, the *system log file* SYSLOG or *TAC queues, asynchronous TACs, USER queues,* SYSOUT/SYSLST or stderr/stdout.
The message destinations for the messages of the UTM tools are SYSOUT/SYSLST and stderr /stdout.

message queue

Queue in which specific messages are kept with transaction management until further processed. A distinction is drawn between *service-controlled queues* and *UTM-controlled queues*, depending on who monitors further processing.

message queuing

Message queuing (MQ) is a form of communication in which the messages are exchanged via intermediate queues rather than directly. The sender and recipient can be separated in space or time. The transfer of the message is independent of whether a network connection is available at the time or not. In openUTM there are *UTM-controlled queues* and *service-controlled queues*.

message segment

The KDCS interface makes it possible to structure messages by means of individual messages segments, which are then transmitted to the communication partner as a whole message.

MSGTAC

Special event service that processes messages with the message destination MSGTAC by means of a program. MSGTAC is an asynchronous service and is created by the operator of the application.

multiplex connection (BS2000 systems)

Special method offered by *OMNIS* to connect terminals to a *UTM application*. A multiplex connection enables several terminals to share a single transport connection.

multi-step service (KDCS)

Service carried out in a number of *dialog steps*.

multi-step transaction

Transaction which comprises more than one *processing step*.

Network File System/Service / NFS

Allows Unix systems to access file systems across the network.

network process (Unix / Linux / Windows systems)

A process in a *UTM application* for connection to the network.

network selector

The network selector identifies a service access point to the network layer of the *OSI reference model* in the local system.

node

Individual computer of a *cluster*.

node application

UTM application that is executed on an individual *node* as part of a *UTM cluster application*.

node bound service

A node bound service belonging to a user can only be continued at the node application at which the user was last signed on. The following services are always node bound:

- Services that have started communications with a job receiver via LU6.1 or OSI TP and for which the job-receiving service has not yet been terminated
- Inserted services in a service stack
- Services that have completed a SESAM transaction

In addition, a user's service is node bound as long as the user is signed-on at a node application.

node filebase

Filename prefix or directory name for the *node application's KDCFILE*, *user log file* and *system log file*.

node recovery

If a node application terminates abnormally and no rapid warm start of the application is possible on its associated *node computer* then it is possible to perform a node recovery for this node on another node in the UTM cluster. In this way, it is possible to release locks resulting from the failed node application in order to prevent unnecessary impairments to the running *UTM cluster application*.

normal termination of a UTM application

Controlled termination of a *UTM application*. Among other things, this means that the administration data in the *KDCFILE* are updated. The *administrator* initiates normal termination (e.g. with KDCSHUT N). After a normal termination, openUTM carries out any subsequent start as a *cold start*.

object identifier

An object identifier is an identifier for objects in an OSI environment which is unique throughout the world. An object identifier comprises a sequence of integers which represent a path in a tree structure.

OMNIS (BS2000 systems)

OMNIS is a “session manager” which lets you set up connections from one terminal to a number of partners in a network concurrently OMNIS also allows you to work with multiplex connections.

online import

In a *UTM cluster application*, online import refers to the import of application data from a normally terminated node application into a running node application.

online update

In a *UTM cluster application,* online update refers to a change to the application configuration or the application program or the use of a new UTM revision level while a *UTM cluster application* is running.

open terminal pool

Terminal pool which is not restricted to clients of a single computer or particular type. Any client for which no computer- or type-specific terminal pool has been generated can connect to this terminal pool.

OpenCPIC

Carrier system for UTM clients that use the *OSI TP* protocol.

OpenCPIC client

OSI TP partner application with the *OpenCPIC* carrier system.

openSM2

The openSM2 product line offers a consistent solution for the enterprise-wide performance management of server and storage systems. openSM2 offers the acquisition of monitoring data, online monitoring and offline evaluation.

openUTM cluster

From the perspective of UPIC clients, **not** from the perspective of the server:Combination of several node applications of a UTM cluster application to form one logical application that is addressed via a common symbolic destination name.

openUTM-D

openUTM-D (openUTM distributed) is a component of openUTM which allows *distributed processing.* openUTM-D is an integral component of openUTM.

OSI-LPAP bundle

LPAP bundle for *OSI TP* partner applications.

OSI-LPAP partner

OSI-LPAP partners are the addresses of the *OSI TP partners* generated in openUTM. In the case of *distributed processing* via the *OSI TP* protocol, an OSI-LPAP partner for each partner application must be configured in the local application. The OSI-LPAP partner represents the partner application in the local application. During communication, the partner application is addressed by the name of the assigned OSI-LPAP partner and not by the application name or address.

OSI reference model

The OSI reference model provides a framework for standardizing communications in open systems. ISO, the International Organization for Standardization, described this model in the ISO IS7498 standard. The OSI reference model divides the necessary functions for system communication into seven logical layers. These layers have clearly defined interfaces to the neighboring layers.

OSI TP

Communication protocol for distributed transaction processing defined by ISO.
OSI TP stands for Open System Interconnection Transaction Processing.

OSI TP partner

Partner of the UTM application that communicates with the UTM application via the OSI TP protocol. Examples of such partners are:

- a UTM application that communicates via OSI TP
- an application in the IBM environment (e.g. CICS) that is connected via openUTM-LU62
- an *OpenCPIC client*
- applications from other TP monitors that support OSI TP

outbound conversation (CPI-C)

See *outgoing conversation*.

outgoing conversation (CPI-C)

A conversation in which the local CPI-C program is the *initiator* is referred to as an outgoing conversation. In the X/Open specification, the term “outbound conversation” is used synonymously with “outgoing conversation”.

page pool

Part of the *KDCFILE* in which user data is stored.
In a *standalone application* this data consists, for example, of *dialog messages*, messages sent to *message queues*, *secondary memory areas*.
In a UTM cluster application, it consists, for example, of messages to *message queues, TLS*.

parameter area

Data structure in which a program unit passes the operands required for a UTM call to openUTM.

partner application

Partner of a UTM application during *distributed processing*. Higher communication protocols are used for distributed processing (*LU6.1*, *OSI TP* or LU6.2 via the openUTM-LU62 gateway).

postselection (BS2000 systems)

Selection of logged UTM events from the SAT logging file which are to be evaluated. Selection is carried out using the SATUT tool.

prepare to commit (PTC)

Specific state of a distributed transaction
Although the end of the distributed transaction has been initiated, the system waits for the partner to confirm the end of the transaction.

preselection (BS2000 systems)

Definition of the UTM events which are to be logged for the *SAT audit*. Preselection is carried out with the UTM-SAT administration functions. A distinction is made between event-specific, user-specific and job-specific (TAC-specific) preselection.

presentation selector

The presentation selector identifies a service access point to the presentation layer of the *OSI reference model* in the local system.

primary storage area

Area in main memory to which the *KDCS program unit* has direct access, e.g. *standard primary working area*, *communication area*.

print administration

Functions for *print control* and the administration of *queued output jobs*, sent to a printer.

print control

openUTM functions for controlling print output.

printer control LTERM

A printer control LTERM allows a client or terminal user to connect to a UTM application. The printers assigned to the printer control LTERM can then be administered from the client program or the terminal. No administration rights are required for these functions.

printer control terminal

This term has been superseded by *printer control LTERM*.

printer group (Unix systems)

For each printer, a Unix system sets up one printer group by default that contains this one printer only. It is also possible to assign several printers to one printer group or to assign one printer to several different printer groups.

printer pool

Several printers assigned to the same *LTERM partner*.

printer process (Unix / Linux 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 / Linux / Windows systems) and tasks (BS2000 systems).

processing step

A processing step starts with the receipt of a *dialog message* sent to the *UTM application* by a *client* or another server application. The processing step ends either when a response is sent, thus also terminating the *dialog step*, or when a dialog message is sent to a third party.

program interface for administration

UTM program interface which helps users to create their own *administration programs*. Among other things, the program interface for administration provides functions for *dynamic configuration*, for modifying properties and application parameters and for querying information on the configuration and the current workload of the application.

program space (BS2000 systems)

Virtual address space of BS2000 which is divided into memory classes and in which both executable programs and pure data are addressed.

program unit

UTM *services* are implemented in the form of one or more program units. The program units are components of the *application program*. Depending on the employed API, they may have to contain KDCS, XATMI or CPIC calls. They can be addressed using *transaction codes*. Several different transaction codes can be assigned to a single program unit.

queue

See *message queue.*

queued output job

Queued output jobs are *asynchronous jobs* which output a message, such as a document, to a printer, a terminal or a transport system application.
Queued output jobs are processed by UTM system functions exclusively, i.e. it is not necessary to create program units to process them.

Quick Start Kit

A sample application supplied with openUTM (Windows systems).

redelivery

Repeated delivery of an *asynchronous message* that could not be processed correctly because, for example, the *transaction* was rolled back or the *asynchronous service* was terminated abnormally. The message is returned to the message queue and can then be read and/or processed again.

reentrant program

Program whose code is not altered when it runs. On BS2000 systems this constitutes a prerequisite for using *shared code*.

request

Request from a *client* or another server for a *service function*.

requestor

In XATMI, the term requestor refers to an application which calls a service.

resource manager

Resource managers (RMs) manage data resources. Database systems are examples of resource managers. openUTM, however, also provides its own resource managers for accessing message queues, local memory areas and logging files, for instance. Applications access RMs via special resource manager interfaces. In the case of database systems, this will generally be SQL and in the case of openUTM RMs, it is the KDCS interface.

restart

See *screen restart.*
see *service restart.*

RFC1006

A protocol defined by the IETF (Internet Engineering Task Force) belonging to the TCP/IP family that implements the ISO transport services (transport class 0) based on TCP/IP.

RSA

Abbreviation for the inventors of the RSA encryption method (Rivest, Shamir and Adleman). This method uses a pair of keys that consists of a public key and a private key. A message is encrypted using the public key, and this message can only be decrypted using the private key. The pair of RSA keys is created by the UTM application.

SAT audit (BS2000 systems)

Audit carried out by the SAT (Security Audit Trail) component of the BS2000 software product SECOS.

screen restart

If a *dialog service* is interrupted, openUTM again displays the *dialog message* of the last completed *transaction* on screen when the service restarts provided that the last transaction output a message on the screen.

SE manager

Web-based graphical user interface (GUI) for the SE series of Business Servers. SE Manager runs on the *management unit* and permits the central operation and administration of server units (with /390 architecture and/or x86 architecture), application units (x86 architecture), net unit and peripherals.

SE server

A Business Server from Fujitsu's SE series.

secondary storage area

Memory area secured by transaction logging and which can be accessed by the KDCS *program unit* with special calls. Local secondary storage areas (LSSBs) are assigned to one *service.* Global secondary storage areas (GSSBs) can be accessed by all services in a *UTM application*. Other secondary storage areas include the *terminal-specific long-term storage (TLS)* and the *user-specific long-term storage (ULS)* .

selector

A selector identifies a service access point to services of one of the layers of the *OSI reference model* in the local system. Each selector is part of the address of the access point.

semaphore (Unix / Linux / Windows systems)

Unix, Linux and Windows systems resource used to control and synchronize processes.

server

A server is an *application* which provides *services* . The computer on which the applications are running is often also referred to as the server.

server-server communication

See *distributed processing*.

server side of a conversation (CPI-C)

This term has been superseded by *acceptor*.

service

Services process the *jobs* that are sent to a server application. A service of a UTM application comprises one or more transactions. The service is called with the *service TAC*. Services can be requested by *clients* or by other servers.

service access point

In the OSI reference model, a layer has access to the services of the layer below at the service access point. In the local system, the service access point is identified by a *selector*. During communication, the *UTM application* links up to a service access point. A connection is established between two service access points.

service chaining (KDCS)

When service chaining is used, a follow-up service is started without a *dialog message* specification after a *dialog service* has completed.

service-controlled queue

Message queue in which the calling and further processing of messages is controlled by *services*. A service must explicitly issue a KDCS call (DGET) to read the message. There are service-controlled queues in openUTM in the variants *USER queue*, *TAC queue* and *temporary queue*.

service restart (KDCS)

If a service is interrupted, e.g. as a result of a terminal user signing off or a *UTM application* being terminated, openUTM carries out a *service restart*. An *asynchronous service* is restarted or execution is continued at the most recent *synchronization point*, and a *dialog service* continues execution at the most recent *synchronization point*. As far as the terminal user is concerned, the service restart for a dialog service appears as a *screen restart* provided that a dialog message was sent to the terminal user at the last synchronization point.

service routine

See *program unit*.

service stacking (KDCS)

A terminal user can interrupt a running *dialog service* and insert a new dialog service. When the inserted *service* has completed, the interrupted service continues.

service TAC (KDCS)

Transaction code used to start a *service* .

session

Communication relationship between two addressable units in the network via the SNA protocol *LU6. 1* .

session selector

The session selector identifies an *access point* in the local system to the services of the session layer of the *OSI reference model*.

shared code (BS2000 systems)

Code which can be shared by several different processes.

shared memory

Virtual memory area which can be accessed by several different processes simultaneously.

shared objects (Unix / Linux / Windows systems)

Parts of the *application program* can be created as shared objects. These objects are linked to the application dynamically and can be replaced during live operation. Shared objects are defined with the KDCDEF statement SHARED-OBJECT.

sign-on check

See *system access control*.

sign-on service (KDCS)

Special *dialog service* for a user in which *program units* control how a user signs on to a UTM application.

single-step service

Dialog service which encompasses precisely one *dialog step*.

single-step transaction

Transaction which encompasses precisely one *dialog step*.

SOA

(Service-Oriented Architecture)
SOA is a system architecture concept in which functions are implemented in the form of re-usable, technically independent, loosely coupled *services*. Services can be called independently of the underlying implementations via interfaces which may possess public and, consequently, trusted specifications. Service interaction is performed via a communication infrastructure made available for this purpose.

SOAP

SOAP (Simple Object Access Protocol) is a protocol used to exchange data between systems and run remote procedure calls. SOAP also makes use of the services provided by other standards, XML for the representation of the data and Internet transport and application layer protocols for message transfer.

socket connection

Transport system connection that uses the socket interface. The socket interface is a standard program interface for communication via TCP/IP.

standalone application

See *standalone UTM application*.

standalone UTM application

Traditional *UTM application* that is not part of a *UTM cluster application*.

standard primary working area (KDCS)

Area in main memory available to all KDCS *program units*. The contents of the area are either undefined or occupied with a fill character when the program unit starts execution.

start format

Format output to a terminal by openUTM when a user has successfully signed on to a *UTM application* (except after a *service restart* and during sign-on via the *sign-on service*).

static configuration

Definition of the *configuration* during generation using the UTM tool *KDCDEF*.

SYSLOG file

See *system log file*.

synchronization point, consistency point

The end of a *transaction*. At this time, all the changes made to the *application information* during the transaction are saved to prevent loss in the event of a crash and are made visible to others. Any locks set during the transaction are released.

system access control

A check carried out by openUTM to determine whether a certain *user ID* is authorized to work with the *UTM application*. The authorization check is not carried out if the UTM application was generated without user IDs.

system log file

File or file generation to which openUTM logs all UTM messages for which SYSLOG has been defined as the *message destination* during execution of a *UTM application.*

TAC

See *transaction code*.

TAC queue

Message queue generated explicitly by means of a KDCDEF statement. A TAC queue is a *service-controlled queue* that can be addressed from any service using the generated name.

temporary queue

Message queue created dynamically by means of a program that can be deleted again by means of a program (see *service-controlled queue*).

terminal-specific long-term storage (KDCS)

Secondary storage area assigned to an *LTERM, LPAP* or *OSI-PAP partner* and which is retained after the application has terminated.

time-driven job

Job which is buffered by openUTM in a *message queue* up to a specific time until it is sent to the recipient. The recipient can be an *asynchronous service* of the same application, a *TAC queue*, a partner application, a terminal or a printer. Time-driven jobs can only be issued by KDCS *program units*.

timer process (Unix / Linux / 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.

TLS termination proxy

A TLS termination proxy is a proxy server that is used to handle incoming TLS connections, decrypting the data and passing on the unencrypted request to other servers.

TNS (Unix / Linux / 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 connection

In the *OSI reference model,* this is a connection between two entities of layer 4 (transport layer).

transport layer security

Transport layer security is a hybrid encryption protocol for secure data transmission in the Internet.

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 access point

See transport system end point.

transport system application

Application which is based directly on a transport system interface (e.g. CMX, DCAM or socket). When transport system applications are connected, the partner type APPLI or SOCKET must be specified during *configuration*. A transport system application cannot be integrated in a *distributed transaction*.

transport system end point

Client/server or server/server communication establishes a connection between two transport system end points. A transport system end point is also referred to as a local application name and is defined using the BCAMAPPL statement or MAX APPLINAME.

TS application

See *transport system application.*

typed buffer (XATMI)

Buffer for exchanging typed and structured data between communication partners. Typed buffers ensure that the structure of the exchanged data is known to both partners implicitly.

UPIC

Carrier system for openUTM clients. UPIC stands for Universal Programming Interface for Communication. The communication with the UTM application is carried out via the *UPIC protocol*.

UPIC Analyzer

Component used to analyze the UPIC communication recorded with *UPIC Capture*. This step is used to prepare the recording for playback using *UPIC Replay*.

UPIC Capture

Used to record communication between UPIC clients and UTM applications so that this can be replayed subsequently (*UPIC Replay*).

UPIC client

The designation for openUTM clients with the UPIC carrier system and for *JConnect clients*.

UPIC protocol

Protocol for the client server communication with *UTM applications*. The UPIC protocol is used by *UPIC clients* and *JConnect clients*.

UPIC Replay

Component used to replay the UPIC communication recorded with *UPIC Capture* and prepared with *UPIC Analyzer*.

user exit

This term has been superseded by *event exit.*

user ID

Identifier for a user defined in the *configuration* for the *UTM application* (with an optional password for *system access control*) and to whom special data access rights (*system access control*) have been assigned. A terminal user must specify this ID (and any password which has been assigned) when signing on to the UTM application. On BS2000 systems, system access control is also possible via *Kerberos*.
For other clients, the specification of a user ID is optional, see also *connection user ID*.
UTM applications can also be generated without user IDs.

user log file

File or file generation to which users write variable-length records with the KDCS LPUT call. The data from the KB header of the *KDCS communication area* is prefixed to every record. The user log file is subject to transaction management by openUTM.

USER queue

Message queue made available to every user ID by openUTM. A USER queue is a *service-controlled queue* and is always assigned to the relevant user ID. You can restrict the access of other UTM users to your own USER queue.

user-specific long-term storage

Secondary storage area assigned to a *user ID*, a *session* or an *association* and which is retained after the application has terminated.

USLOG file

See *user log file.*

UTM application

A UTM application provides *services* which process jobs from *clients* or other applications. openUTM is responsible for transaction logging and for managing the communication and system resources. From a technical point of view, a UTM application is a process group which forms a logical server unit at runtime.

UTM client

See *client.*

UTM cluster application

UTM application that has been generated for use on a cluster and that can be viewed logically as a **single** application.
In physical terms, a UTM cluster application is made up of several identically generated UTM applications running on the individual cluster *nodes*.

UTM cluster files

Blanket term for all the files that are required for the execution of a UTM cluster application on Unix, Linux and Windows systems. This includes the following files:

- *Cluster configuration file*
- *Cluster user file*
- Files belonging to the *cluster page pool*
- *Cluster GSSB file*
- *Cluster ULS file*
- Files belonging to the *cluster administration journal**
- *Cluster lock file**
- Lock file for start serialization*

The files indicated by * are created when the first node application is started. All the other files are created on generation using KDCDEF.

UTM-controlled queue

Message queues in which the calling and further processing of messages is entirely under the control of openUTM. See also *asynchronous job, background job* and *asynchronous message.*

UTM-D

See *openUTM-D.*

UTM-F

UTM applications can be generated as UTM-F applications (UTM fast). In the case of UTM-F applications, input from and output to hard disk is avoided in order to increase performance. This affects input and output which *UTM-S* uses to save user data and transaction data. Only changes to the administration data are saved.
In UTM cluster applications that are generated as UTM-F applications (APPLI-MODE=FAST), application data that is valid throughout the cluster is also saved. In this case, GSSB and ULS data is treated in exactly the same way as in UTM cluster applications generated with UTM-S. However, service data relating to users with RESTART=YES is written only when the relevant user signs off and not at the end of each transaction.

UTM generation

Static configuration of a *UTM application* using the UTM tool KDCDEF and creation of an application program.

UTM message

Messages are issued to *UTM message destinations* by the openUTM transaction monitor or by UTM tools (such as *KDCDEF*). A message comprises a message number and a message text, which can contain *inserts* with current values. Depending on the message destination, either the entire message is output or only certain parts of the message, such as the inserts).

UTM page

A UTM page is a unit of storage with a size of either 2K, 4K or 8 K. In *standalone UTM applications,* the size of a UTM page on generation of the UTM application can be set to 2K, 4K or 8 K. The size of a UTM page in a *UTM cluster application* is always 4K or 8 K. The *page pool* and the restart area for the KDCFILE and *UTM cluster files* are divided into units of the size of a UTM page.

utmpath (Unix / Linux / Windows systems)

The directory under which the openUTM components are installed is referred to as *utmpath* in this manual.
To ensure that openUTM runs correctly, the environment variable UTMPATH must be set to the value of *utmpath*. On Unix and Linux systems, you must set UTMPATH before a UTM application is started. On Windows systems UTMPATH is set in accordance with the UTM version installed most recently.

UTM-S

In the case of UTM-S applications, openUTM saves all user data as well as the administration data beyond the end of an application and any system crash which may occur. In addition, UTM-S guarantees the security and consistency of the application data in the event of any malfunction. UTM applications are usually generated as UTM-S applications (UTM secure).

UTM SAT administration (BS2000 systems)

UTM SAT administration functions control which UTM events relevant to security which occur during operation of a *UTM application* are to be logged by *SAT*. Special authorization is required for UTM SAT administration.

UTM socket protocol (USP)

Proprietary openUTM protocol above TCP/IP for the transformation of the Socket interface received byte streams in messages.

UTM system process

UTM process that is started in addition to the processes specified via the start parameters and which only handles selected jobs. UTM system processes ensure that UTM applications continue to be reactive even under very high loads.

UTM terminal

This term has been superseded by *LTERM partner*.

UTM tool

Program which is provided together with openUTM and which is needed for UTM specific tasks (e.g for configuring).

virtual connection

Assignment of two communication partners.

warm start

Start of a *UTM-S* application after it has terminated abnormally. The *application information* is reset to the most recent consistent state. Interrupted *dialog services* are rolled back to the most recent *synchronization point*, allowing processing to be resumed in a consistent state from this point (*service restart*). Interrupted *asynchronous services* are rolled back and restarted or restarted at the most recent *synchronization point*.
For *UTM-F* applications, only configuration data which has been dynamically changed is rolled back to the most recent consistent state after a restart due to a preceding abnormal termination.
In UTM cluster applications, the global locks applied to GSSB and ULS on abnormal termination of this node application are released. In addition, users who were signed on at this node application when the abnormal termination occurred are signed off.

WebAdmin

Web-based tool for the administration of openUTM applications via a Web browser. WebAdmin includes not only the full function scope of the *administration program interface* but also additional functions.

Web service

Application which runs on a Web server and is (publicly) available via a standardized, programmable interface. Web services technology makes it possible to make UTM program units available for modern Web client applications independently of the programming language in which they were developed.

WinAdmin

Java-based tool for the administration of openUTM applications via a graphical user interface. WinAdmin includes not only the full function scope of the *administration program interface* but also additional functions.

work process (Unix / Linux / Windows systems)

A process within which the *services* of a *UTM application* run.

workload capture & replay

Family of programs used to simulate load situations; consisting of the main components *UPIC Capture*, *UPIC Analyzer* and *Upic Replay* and - on Unix, Linux and Windows systems - the utility program *kdcsort*. Workload Capture & Replay can be used to record UPIC sessions with UTM applications, analyze these and then play them back with modified load parameters.

WS4UTM

WS4UTM (**W**eb**S**ervices for open**UTM**) provides you with a convenient way of making a service of a UTM application available as a Web service.

XATMI

XATMI (X/Open Application Transaction Manager Interface) is a program interface standardized by X /Open for program-program communication in open networks.
The XATMI interface implemented in openUTM complies with X/Open's XATMI CAE Specification. The interface is available in COBOL and C. In openUTM, XATMI can communicate via the OSI TP, *LU6.1* and UPIC protocols.

XHCS (BS2000 systems)

XHCS (Extended Host Code Support) is a BS2000 software product providing support for international character sets.

XML

XML (eXtensible Markup Language) is a metalanguage standardized by the W3C (WWW Consortium) in which the interchange formats for data and the associated information can be defined.

10 Abbreviations

Please note: Some of the abbreviations used here derive from the German acronyms used in the original German product(s).

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AES	Advanced Encryption Standard
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder - Loader - Starter (BS2000 systems)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Coded Character Set
CCSN	Coded Character Set Name
CICS	Customer Information Control System
CID	Control Identification
CMX	Communication Manager in Unix, Linux and Windows Systems
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000 systems)
DB	Database
DBH	Database Handler
DC	Data Communication
DCAM	Data Communication Access Method

DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000 systems)
DMS	Data Management System
DNS	Domain Name Service
DP	Distribted Processing
DSS	Terminal (Datensichtstation)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans™
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GCM	Galois/Counter Mode
GSSB	Global Secondary Storage Area
HIPLEX®	Highly Integrated System Complex (BS2000 systems)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IFG	Interactive Format Generator
ILCS	Inter-Language Communication Services (BS2000 systems)
IMS	Information Management System (IBM)
IPC	Inter-Process Communication
IRV	International Reference Version
ISO	International Organization for Standardization
Java EE	Java Platform, Enterprise Edition
JCA	Java EE Connector Architecture
JDK	Java Development Kit

KAA	KDCS Application Area
KB	Communication Area
KBPRG	KB Program Area
KDCADMI	KDC Administration Interface
KDCS	Compatible Data Communication Interface
KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000 systems)
LSSB	Local Secondary Storage Area
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000 systems)
NB	Message Area
NEA	Network Architecture for BS2000 Systems
NFS	Network File System/Service
NLS	Native Language Support
OLTP	Online Transaction Processing
OML	Object Module Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Process Identification
PIN	Personal Identification Number
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Computer Center Accounting Procedure
RDF	Resource Definition File

RM	Resource Manager
RSA	Encryption algorithm according to Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000 systems)
RTS	Runtime System
SAT	Security Audit Trail (BS2000 systems)
SECOS	Security Control System
SEM	SE Manager
SGML	Standard Generalized Markup Language
SLU	Secondary Logical Unit
SM2	Software Monitor 2
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primary Working Area
SQL	Structured Query Language
SSB	Secondary Storage Area
SSL	Secure Socket Layer
SSO	Single Sign-On
TAC	Transaction Code
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-Specific Long-Term Storage
TLS	Transport Layer Security
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaction Mode)
TPR	Privileged Function State in BS2000 systems (Task Privileged)
TPSU	Transaction Protocol Service User

TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Non-Privileged Function State in BS2000 systems (Task User)
TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universal Database System
UDT	Unstructured Data Transfer
ULS	User-Specific Long-Term Storage
UPIC	Universal Programming Interface for Communication
USP	UTM Socket Protocol
UTM	Universal Transaction Monitor
UTM-D	UTM Variant for Distributed Processing in BS2000 systems
UTM-F	UTM Fast Variant
UTM-S	UTM Secure Variant
UTM-XML	openUTM XML Interface
VGID	Service ID
VTSU	Virtual Terminal Support
WAN	Wide Area Network
WS4UTM	Web-Services for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (X/Open interface for acess to the resource manager)
XAP	X/OPEN ACSE/Presentation programming interface
XAP-TP	X/OPEN ACSE/Presentation programming interface Transaction Processing extension
XATMI	X/Open Application Transaction Manager Interface
XCS	Cross Coupled System
XHCS	eXtended Host Code Support
XML	eXtensible Markup Language

11 Related publications

You will find the manuals on the internet at https://bs2manuals.ts.fujitsu.com.

openUTM documentation

openUTMConcepts and Functions

User Guide

openUTM Programming Applications with KDCS for COBOL, C and C++

Core Manual

openUTM Generating Applications

User Guide

openUTM Using UTM Applications on BS2000 Systems

User Guide

openUTM Using UTM Applications on Unix, Linux and Windows Systems

User Guide

openUTM Administering Applications

User Guide

openUTM Messages, Debugging and Diagnostics on BS2000 Systems

User Guide

openUTM Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems

User Guide

openUTM Creating Applications with X/Open Interfaces

User Guide

openUTM XML for openUTM

openUTM Client (Unix systems) for the OpenCPIC Carrier System
Client-Server Communication with openUTM

User Guide

openUTM Client for the UPIC Carrier System
Client-Server Communication with openUTM

User Guide

openUTM WinAdmin
Graphical Administration Workstation for openUTM

Description and online help system

openUTM WebAdmin
Web Interface for Administering openUTM

Description and online help system

openUTM, openUTM-LU62
Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications

User Guide

openUTM (BS2000)
Programming Applications with KDCS for Assembler
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for Fortran
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for Pascal-XT
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for PL/I
Supplement to Core Manual

WS4UTM (Unix systems and Windows systems**)**
WebServices for openUTM

Documentation for the openSEAS product environment

BeanConnect

User Guide

openUTM-JConnect
Connecting Java Clients to openUTM

User documentation and Java docs

WebTransactions
Concepts and Functions

WebTransactions
Template Language

WebTransactions
Web Access to openUTM Applications via UPIC

WebTransactions
Web Access to MVS Applications

WebTransactions
Web Access to OSD Applications

Documentation for the BS2000 environment

AID Advanced Interactive Debugger
Core Manual

User Guide

AID Advanced Interactive Debugger
Debugging of COBOL Programs

User Guide

AID Advanced Interactive Debugger
Debugging of C/C++ Programs

User Guide

BCAM
BCAM Volume 1/2

User Guide

BINDER
User Guide

BS2000 OSD/BC
Commands Volume 1 - 7

User Guide

BS2000 OSD/BC
Executive Macros

User Guide

BS2IDE
Eclipse-based Integrated Development Environment for BS2000
User Guide and Installation Guide
Web page: *https://bs2000.ts.fujitsu.com/bs2ide/*

BLSSERV
Dynamic Binder Loader / Starter in BS2000/OSD
User Guide

DCAM
COBOL Calls
User Guide

DCAM
Macros
User Guide

DCAM
Program Interfaces
Description

FHS
Format Handling System for openUTM, TIAM, DCAM
User Guide

IFG for FHS
User Guide

HIPLEX AF
High-Availability of Applications in BS2000/OSD
Product Manual

HIPLEX MSCF
BS2000 Processor Networks
User Guide

IMON
Installation Monitor
User Guide

MT9750 (MS Windows)
9750 Emulation under Windows
Product Manual

OMNIS/OMNIS-MENU
Functions and Commands
User Guide

OMNIS/OMNIS-MENU
Administration and Programming

User Guide

OSS (BS2000)
OSI Session Service
User Guide

openSM2
Software Monitor

User Guide

RSO
Remote SPOOL Output

User Guide

SECOS
Security Control System

User Guide

SECOS
Security Control System

Ready Reference

SESAM/SQL
Database Operation

User Guide

TIAM
User Guide

UDS/SQL
Database Operation

User Guide

Unicode in BS2000/OSD
Introduction

VTSU
Virtual Terminal Support

User Guide

XHCS
8-Bit Code and Unicode Support in BS2000/OSD

User Guide

Documentation for the Unix, Linux and Windows system environment

CMX V6.0 (Unix systems)
Betrieb und Administration (only available in German)
User Guide

CMX V6.0
Programming CMX Applications
Programming Guide

OSS (UNIX)
OSI Session Service
User Guide

PRIMECLUSTER™
Concepts Guide (Solaris, Linux)

openSM2

The documentation of openSM2 is provided in the form of detailed online help systems, which are delivered with the product.

Other publications

CPI-C

X/Open CAE Specification
Distributed Transaction Processing:
The CPI-C Specification, Version 2
ISBN 1 85912 135 7

Reference Model
X/Open Guide
Distributed Transaction Processing:
Reference Model, Version 2
ISBN 1 85912 019 9

REST
Architectural Styles and the Design of Network-based Software Architectures
Dissertation Roy Fielding

TX
X/Open CAE Specification
Distributed Transaction Processing:
The TX (Transaction Demarcation) Specification
ISBN 1 85912 094 6

XATMI
X/Open CAE Secification
Distributed Transaction Processing

The XATMI Specification
ISBN 1 85912 130 6

XML
W3C specification (www consortium)

Web page: *http://www.w3org/XML*