English

Fujitsu Software BS2000

openUTM V7.0

Concepts and Functions

User Guide

Valid for:
openUTM V7.0A00

Edition November 2019

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

Table of Contents

Concepts and Functions

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.

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.

Brief overviews are given of the features of BeanConnect and WebTransactions. There are separate manuals available for these products.

1.1 Summary of contents and target group

This manual, “Concepts and Functions” introduces you to the openUTM product family and enables you to get started working with openUTM. It is aimed particularly at those who are not yet familiar with openUTM. But even if you already know and work with openUTM, you will find the manual useful for the overview it provides of the range of functions and capabilities of the product.

Rather than being concerned with the syntactic subtleties of individual statements or with the details of specific interfaces, the manual provides a general overview of the features and possible applications of openUTM. Equipped with this information, you will have no problem understanding the other manuals in the openUTM series.

Chapter 2 contains a brief description of the features of openUTM, some of which are dealt with in further detail in chapters 3 through 13.

openUTM is available for all established Unix platforms and Windows platforms and for BS2000 systems. Its functionality and interfaces are to a large extent non-platform-specific. The information given in the first chapters therefore applies for all platforms.

The last three chapters contain some platform-specific information. Chapter 14 deals with BS2000 systems, chapter 15 covers all Unix platforms and chapter 16 covers Windows platforms. The detailed lists at the end of the manual - the glossary, the list of abbreviations, the list of related publications, and the index - will help you to find your way around this manual.

Obviously, the manual cannot answer all your questions, but it will point you in the right direction for finding solutions to specific problems:

This symbol is used to refer you to more detailed information on the relevant Topic.

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

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

1.2 Summary of contents of the openUTM documentation

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

1.2.1 openUTM documentation

The openUTM documentation consists of manuals, 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 are available on the internet at https://bs2manuals.ts.fujitsu.com. For the BS2000 platform, you will also find the manuals on the Softbook DVD.

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

1.2.2 Documentation for the openSEAS product environment

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

Integrating Java EE application servers and UTM applications

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

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

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.

1.2.3 Readme files

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

Readme files are available to you online in addition to the product manuals under the various products at https://bs2manuals.ts.fujitsu.com. For the BS2000 platform, you will also find the Readme files on the Softbook DVD.

Information on BS2000 systems

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

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

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

Additional product information

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

1.3 Changes in openUTM V7.0

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

1.3.1 New server functions

UTM as HTTP-Server

A UTM application can also act as an HTTP server.

GET, PUT, POST and DELETE are supported as methods. In addition to HTTP, access via HTTPS is also supported.

The following interfaces have been changed:

- Generation

  All systems:

  - KDCDEF statement BCAMAPPL:
    - Additional specification for the transport protocol for the operand T-PROT= with value SOCKET

      *USP: The UTM socket protocol is to be used on connections from this access point.

      *HTTP: The HTTP protocol is to be used for connections from this access point.

      *ANY: Both the UTM socket protocol and the HTTP protocol are supported on connections from this access point.

    - Additional specification for encryption for the operand T-PROT= with value SOCKET

      SECURE: On connections from this access point, communication takes place using transport layer security (TLS).

    - New operand USER-AUTH = *NONE | *BASIC. Herewith you can specify which authentication mechanism HTTP clients must use for this access point.
    - KDCDEF statement HTTP-DESCRIPTOR:
      This statement defines a mapping of the path received in an HTTP request to a TAC and additional processing parameters can be specified.

  BS2000 systems:

  - KDCDEF statement CHAR-SET:
    With this statement, each of the four UTM code conversions provided by openUTM can be assigned up to four character set names.

- Programming

  - KDCS communication area (KB):
    In the header of the KDCS communication area, there are new indicators for the client protocols HTTP, USP-SECURE, and HTTPS in the *kccp/KCCP* field.
  - KDCS call INIT PU:
    - The version of the interface has been increased to 7.
    - To obtain the complete available information, the value 372 must be specified in the KCLI field.
    - New fields for requesting (KCHTTP/http_info) and returning (KCHTTPINF/httpInfo) HTTP-specific information.

- Administration interface KDCADMI
  - The data structure version of KDCADMI has been changed to version 11 (field *version_data* in the parameter area).

- New structure *kc_http_descriptor_str* in the identification area to support the HTTP descriptor.
- New structure *kc_character_set_str* in the identification area for supporting the HTTP character set.
- New fields *secure_soc* and *user_auth* in structure *kc_bcamappl_str* for the support of HTTP access points.

- UTM-HTTP program interface

  In addition to the KDCS interface, UTM provides an interface for reading and writing HTTP protocol information and handling the HTTP message body.
  The functions of the interface are briefly listed below:

  - Function *kcHttpGetHeaderByIndex()*
    This function returns the name and value of the HTTP header field for the specified index.
  - Function *kcHttpGetHeaderByName()*
    The function returns the value of the HTTP header field specified by the name.
  - Function *kcHttpGetHeaderCount()*
    This function returns the number of header fields contained in the HTTP request, that can be read by the program unit.
  - Function *kcHttpGetMethod()*
    This function returns the HTTP method of the HTTP request.
  - Function *kcHttpGetMputMsg()*
    This function returns the MPUT message generated by the program unit.
  - Function *kcHttpGetPath()*
    This function returns the HTTP path of the HTTP request normalized with KC_HTTP_NORM_UNRESERVED.
  - Function *kcHttpGetQuery()*
    This function returns the HTTP query of the HTTP request normal ized with KC_HTTP_NORM_UNRESERVED.
  - Function *kcHttpGetRc2String()*
    Help function to convert a function result of type enum into a printable zero terminated string.
  - Function *kcHttpGetReqMsgBody()*
    This function returns the message body of the HTTP request.
  - Function *kcHttpGetScheme()*
    This function returns the schema of the HTTP request.
  - Function *kcHttpGetVersion()*
    This function returns the version of the HTTP request.
  - Function *kcHttpPercentDecode()*
    Function to convert characters in percent representation in strings to their normal one-character representation.
  - Function *kcHttpPutHeader()*
    This function passes an HTTP header for the HTTP response.
  - Function *kcHttpPutMgetMsg()*
    This function passes a message for the program unit, which can be read with MGET.
  - Function *kcHttpPutRspMsgBody()*
    This function passes a message for the message body of the HTTP response.
  - Function *kcHttpPutStatus()*
    This function passes a HTTP status code for the HTTP response.

- Communication via the Secure Socket Layer (SSL)
  BS2000 systems:

  - If a BCAMAPPL with T-PROT=(SOCKET,...,SECURE) has been generated for a UTM application, an additional task is started with a reverse proxy when UTM starts the application. The reverse proxy acts as the TLS Termination Proxy for the application and handles all SSL communication.

  Unix, Linux and Windows systems :

  - Another network process of the type *utmnetssl* is available for secure access with TLS.

    If BCAMAPPL is generated with T-PROT=(SOCKET,...,SECURE) for a UTM application, a number of *utmnetssl* processes are started when UTM is started. The number of these processes depends on the value LISTENER-ID of these BCAMAPPL objects. All TLS communication for the assigned BCAMAPPL port numbers is handled in a *utmnetssl* process.

Encryption

The encryption functionality in UTM between a UTM application and a UPIC client has been revised. Security gaps have been closed, modern methods have been adopted and delivery has been simplified as follows:

- UTM-CRYPT variant

  Previously, the encryption functionality in UTM was only available if the product UTM-CRYPT had been installed. With UTM V7.0 this is no longer necessary. As of this version, the decision as to whether or not to use the encryption functionality is made via generation or at the time of application start.
- Security
  A vulnerability has been fixed in the communication between a UTM application and a UPIC client.

  > ! This means that encrypted communication with a UTM application V7.0 is only possible together with UPIC client applications as of UPIC V7.0!

- Encryption Level 5 (*Unix, Linux and Windows systems)*

  KDCDEF statements PTERM, TAC and TPOOL
  The operand ENCRYPTION-LEVEL has an additional level 5, where the Diffie-Hellman method based on Elliptic Curves is used to agree the session key and input/output messages are encrypted with the AES-GCM algorithm.

OSI-TP communication and port numbers

BS2000 systems:

- KDCDEF statement OSI-CON
  The operand LISTENER-PORT can also be specified on BS2000 systems.
- Administration interface KDCADMI
  In the structure *kc_osi_con_str*, the port number is also displayed in the *listener-port* field on BS2000 systems.

Subnets

In a UTM application, subnets can also be generated on BS2000 systems in order to restrict access to UTM applications to defined IP address ranges. In addition, name resolution can be controlled via DNS.

The following interfaces have been changed for this purpose:

- Generation
  BS2000 systems:
  - KDCDEF statement SUBNET:
    The SUBNET statement can also be specified on BS2000 systems.

  All systems:
  - KDCDEF statement SUBNET:

    RESOLVE-NAMES=YES/NO can be used to specify whether or not a name resolution via DNS is to take place after a connection is established.

    If name resolution takes place, the real processor name of the communication partner is displayed via the administration interface and in messages. Otherwise, the IP address of the communication partner and the name of the subnet defined in the generation are displayed as the processor name.
- Administration interface KDCADMI
  The structures *kc_subnet_str* and kc_tpool_str contain a new field *resolve_names*.

Access data for the XA database connection

A modified but not yet activated user name for the XA database connection can be read by Administration (KDCADMI):

- Operation code KC_GET_OBJECT:
  Data Structure *kc_db_info_str*: New field *db_new_userid*.

Reconnect for the XA database connection

If an XA action to control the transaction detects that the connection to the database has been lost, the system tries to renew the connection and repeat the XA action.

Only if this is not successful, the affected UTM process and the UTM application are terminated abnormally. Previously, the UTM application was terminated abnormally, if a XA-Connection was lost without trying to reconnect.

Other changes

- XA messages
  The messages regarding the XA interface were extended by the inserts UTM-Userid and TAC. The messages K204-K207, K212-K215 and K217-K218 are affected.
- UTM-Tool KDCEVAL
  In the TRACE 2 record of KDCEVAL the type of the last order (bourse announcement) was recorded in the WAITEND record (first two bytes can be printed).

1.3.2 Discontinued server functions

In particular, the following functions has been discontinued:

- KDCDEF utiliy
  Several functions have been deleted and can no longer be generated in KDCDEF. If they are still specified, this will be rejected with a syntax error in the KDCDEF run.
  - KDCDEF statement PTERM
    Operand values 1 and 2 for ENCRYPTION-LEVEL
  - KDCDEF statement TPOOL
    Operanden values 1 and 2 for ENCRYPTION-LEVEL
  - KDCDEF statement TAC
    Operanden value 1 for ENCRYPTION-LEVEL
- *BS2000 systems*
  - UTM Cluster:
    UTM cluster applications are no longer supported on BS2000 systems.
- *Unix, Linux and Windows systems*
  - TNS operation:
    When starting a UTM application, the TNS generation is no longer read. The addressing information must be stored completely during configuration with KDCDEF.

1.3.3 New client functions

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.

1.3.4 New functions for openUTM WinAdmin

WinAdmin supports all new features of openUTM 7.0 relating to the program interface for the administration.

1.3.5 New functions for openUTM WebAdmin

WebAdmin supports all new features of openUTM 7.0 relating to the program interface for the administration.

2 Overview of openUTM features

This chapter provides an overview of the most important functions of the openUTM application server and examines the IT environment in which openUTM is typically used.

2.1 openUTM - the “High-end Transaction Processing Platform”

As a high-end transaction processing platform, openUTM is used for OLTP (online transaction processing) applications. These may be banking applications or travel booking systems, for example: bank customers transfer payments between accounts at different banks in different locations – and not just by actually going to the bank. They may also do it by phone or via the World Wide Web. Travel agents book international flights and hotel rooms and have online access to the databases of airlines and hotel chains. openUTM is also the basis for enterprise-wide, integrated IT solutions based on client/server concepts or for stockkeeping and production control systems.

openUTM can handle dialog-controlled services or asynchronous services that are detached from the dialog. Its message queuing features support workflow concepts, mobile computing and similar applications.

As an application server, openUTM takes on the following tasks:

- openUTM forms the runtime environment for service routines.
- The service routines can be generated statically or added dynamically to the application during live operation.
- The service routines use standardized interfaces to the openUTM application server (X/Open: CPI-C, TX, XATMI; DIN: KDCS, UTM HTTP).
- The service routine can accesses databases directly (by means of SQL).
- The openUTM application server coordinates the transactions with the database system.
- openUTM controls the exchange of information between clients, applications and resources on the basis of transactions. openUTM thus ensures reliability, availability and performance – even in the case of complex distributed structures.

openUTM provides the KDCADMI programming interface which can be used to administer UTM applications.
For communication with HTTP-clients openUTM provides the UTM-HTTP programming interface.

UTM applications can be operated both in the form of a standalone application (one host) and in the form of a UTM cluster application (on more than one host). Cluster functionality is available for Linux, Unix and Windows systems.

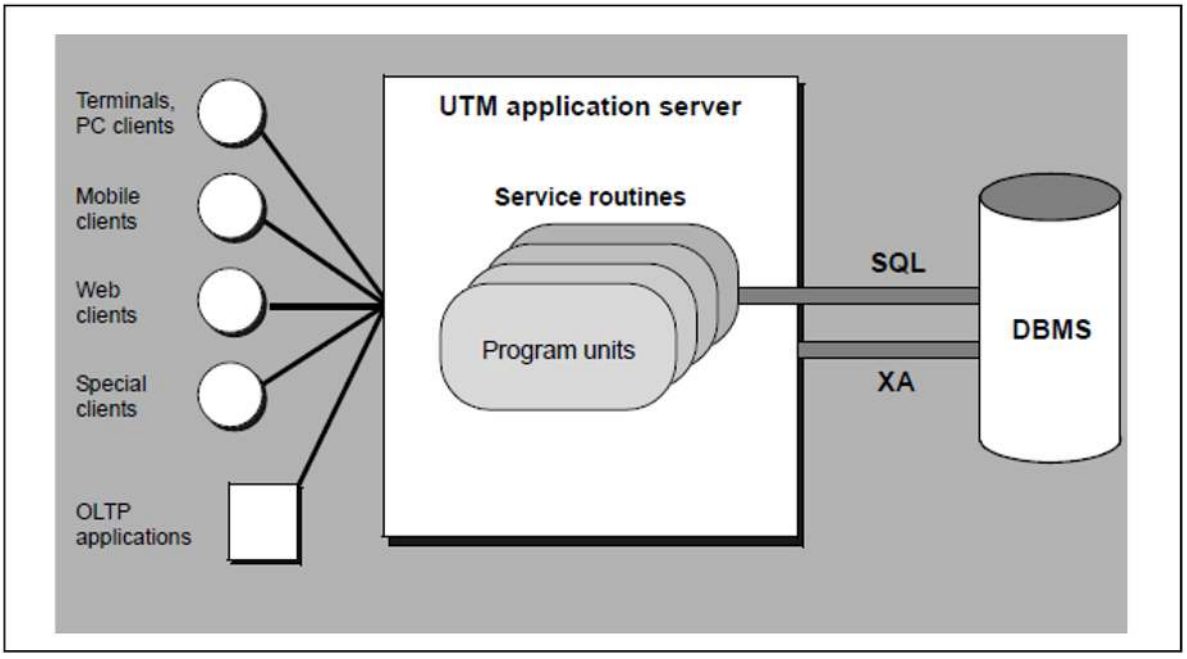


Figure 1: The UTM application server as a high-end transaction processing platform

openUTM supports mult-tier architectures that permit distribution of business logic (processing) to a number of servers. openUTM acts as a distributed, high-level operating system.

When designing applications, there is a range of powerful functions on which you can build: openUTM controls global transactions, optimizes the utilization of system resources (working memory, CPU, etc.), manages concurrent access and takes care of access control, the establishment of network connections and a great deal more.

The specifics of heterogeneous networks and different platforms do not find expression in the application components, so the various components remain environment-independent and offer great flexibility in that they can be replaced easily and used universally.

You will find detailed information on client/server computing and server-to-server communication with openUTM in a separate chapter of the manual devoted to this subject (chapter “Integration scenarios with openUTM”).

2.2 Transaction concept and restart functions

One of the most important tasks of openUTM is to ensure the consistency and integrity of application data, even in the event of problems such as power failures or system crashes. All processes under openUTM are therefore based on the transaction concept.

A transaction is a collection of operational steps with certain properties, known as ACID properties:

- **A**tomicity
  The steps combined within a transaction form an atomic unit: either all the steps within the transaction are performed, or none of them are performed (all-or-nothing rule). If a transaction is not executed in full for some reason, it is rolled back, i.e. all data is reset to its status before the transaction was started.
- **C**onsistency
  The individual steps are performed correctly. If the resources (databases, printers, message queues, etc.) were in an inconsistent state before the transaction was started, they will be restored to a consistent state after the transaction has ended.
- **I**solation
  If several transactions are to be executed in parallel, concurrent changes are carried out as if the transactions were executed consecutively, i.e. with no overlap. Other transactions are not aware of any intermediate states, as the transactions are isolated from each other.
- **D**urability
  Once a transaction has been concluded successfully, the changes made are permanent, i.e. they are retained even after a system crash. All modifications are logged at the end of the transaction. The end of a transaction therefore also represents a synchronization point.

For example, a credit transfer consisting of the steps “debit amount” and “credit amount” fulfills the atomicity condition provided each debit operation is followed by a credit operation. It fulfills the consistency condition if the amount is debited and credited in the prescribed format and at the right location. It fulfills the isolation condition if the account balances are read-/write-protected during processing. Finally, it fulfills the durability condition if the updates are not lost following a system error, e.g. due to a power failure.

openUTM fully integrates all resources into transaction management, and can thus offer effective automatic restart functions.

In the event of an automatic restart, openUTM resynchronizes not only the data in the databases with the data in the application, but also any interrupted services, local resources (e.g. storage areas), queues, communication links, and front-ends such as terminals, PCs, workstations, and printers.

For instance, the restart functionality allows all client PCs to continue operating with the last consistent PC screen, even after a system crash (PCs, network, server).

You will find additional information on the subject of “restarts” in this manual in section “The openUTM restart functions”.

2.3 Coordinating with databases and resource managers

One of the most important functions of a transaction monitor is to coordinate with resource managers (the term “resource manager” is explained in section "X/Open conformance of openUTM"). Since openUTM supports the XA interface standardized by X/Open, it can interact with all resource managers that offer this interface, even if these are mixed within a transaction. More than one resource manager can be connected to a given UTM application at the same time.

Database systems play a central role among the resource managers. This section therefore concentrates on coordination with database systems. The database systems with which a UTM application is to interact are defined when generating the UTM application.

Supported database systems

openUTM (BS2000) supports coordination with the following database systems:

- UDS/SQL
- SESAM/SQL
- Oracle
- LEASY (from the point of view of openUTM, the LEASY file system behaves as a database system)

openUTM for Unix, Linux and Windows systems supports coordination with the following database system:

- Oracle

Coordination between UTM transactions and database transactions

The data records stored in a database are subject to the locking and synchronization mechanisms of the respective database system, and not those of openUTM. To ensure global data consistency, UTM transactions must be synchronized with those of all database systems involved. For this purpose, openUTM uses the **two-phase commit** protocol. As soon as processing of a local transaction is complete, the transaction is initially switched to

the “preliminary end of transaction (PET)” state. The global end of transaction (commit) is not set until all the transactions involved have reached the PET state. Figure 2 illustrates this procedure by means of a simple example.

The UTM transaction and the database transactions can therefore be regarded as subsets of a global transaction. The global transaction cannot be terminated successfully until all of its subsets have been completed. If one of the subsets- either a UTM transaction or a database transaction - cannot be concluded, the global transaction is rolled back. This means that all changes made in the databases and the UTM storage areas by the transaction are reversed, and the asynchronous and output services called are canceled.

As far as the programmer is concerned, there is no additional effort involved in coordinating between openUTM and database systems.

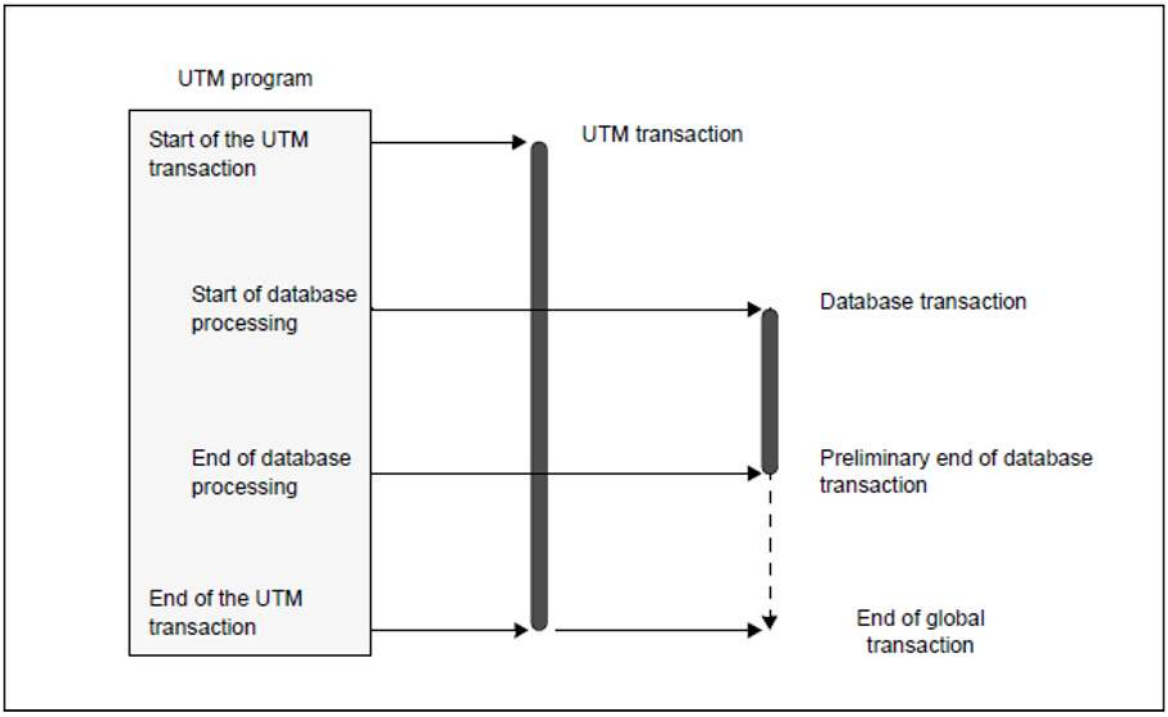


Figure 2: Coordination between UTM and database transactions based on the two-phase commit protocol

Coordination with distributed and heterogeneous databases

All transaction systems support coordination with a resource manager, but only openUTM offers the following additional options:

A UTM application can coordinate with several database systems, i.e. a **single** UTM service program can contain calls to **numerous** database systems. In the context of distributed processing, data from several different database systems can be processed on a number of systems within the same transaction. openUTM thus supports secure interaction with distributed and heterogeneous databases on all conventional hardware and operating system platforms - even within the same service program.

Thanks to these features, openUTM is much more flexible than other transaction monitors: it allows you to integrate the existing data logistics into UTM applications without modification, and offers greater scope when extending the IT environment.

Error handling and diagnostics when coordinating with database systems

If errors occur when coordinating with database systems, openUTM performs all error handling procedures. There is no need for the programmer to take any special precautions.

The database system informs openUTM whether or not a call can be executed successfully. If an error occurs, openUTM checks the error level and reacts accordingly: it rolls back the transactions to the last synchronization point and outputs error messages containing information on the cause of the error.

To simplify diagnostics, an entry is created in an area of the UTM dump for each database call. A special tool is also provided for simple, precise analysis of the UTM dump.

Support for failover with Oracle Real Application Clusters

When working with Oracle Real Application Clusters, openUTM is able to resume the application run normally in case of a failover. Transactions interrupted by the failover are terminated correctly when possible. If this is not possible, then they are rolled back by openUTM or by the database system so that the database is also consistent after the failover.

Internal interface between openUTM and database systems

openUTM controls interaction with database systems via a standard, uniform interface. It is thus completely independent of any implementation-related specifications of the different database systems.
The interface used is the XA interface standardized by X/Open, and on BS2000 systems the functionally equivalent interface IUTMDB is supported.

Support for other resource managers

Database systems are only one type of resource manager. UTM transaction management also covers transaction-oriented file systems (e.g. LEASY), message queues, local storage areas, log files, and network connections. Regardless of whether the resource manager is a UTM resource manager (UTM message queues, local storage areas, log files) or an external resource manager (non-Siemens message queueing system), openUTM coordinates transaction management across all applications, operating Systems and hardware platforms.

2.4 UTM cluster application on Linux, UNIX or Windows Systems

Cluster support on Linux, UNIX and Windows Systems is a key function in openUTM.

A cluster is a number of computers (nodes) connected over a fast network and which share common peripherals. A UTM application can run as a UTM cluster application on a cluster.

In the same way as for standalone applications, UTM cluster applications provide you with the full functionality of openUTM.

UTM cluster application

Unlike a standalone UTM application, a UTM cluster application consists of a number of identically configured UTM applications which run on a node of a cluster and can be understood logically to be a single application. Each of these so-called node applications runs on a separate node.

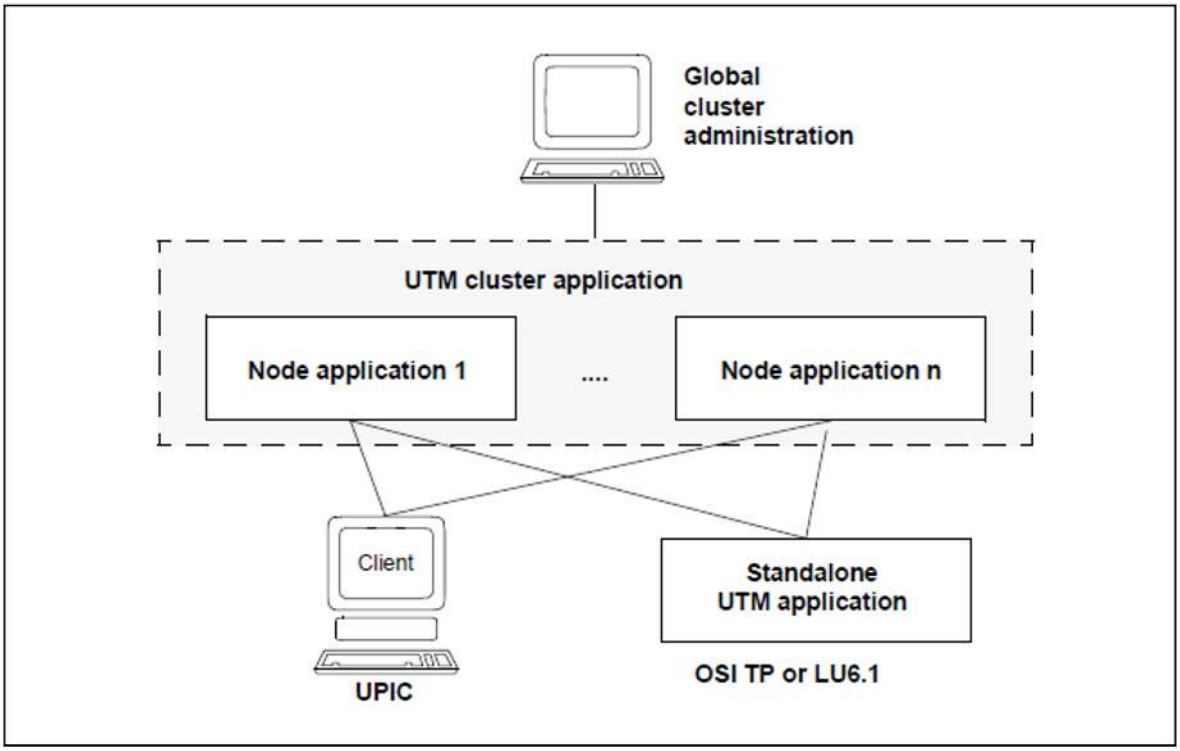


Figure 3: UTM cluster application

On Unix, Linux or Windows Systems UTM cluster applications can be distributed over up to 32 nodes.

UTM cluster applications offer advantages in terms of load distribution and high availability:

- Essential high-availability functions such as application monitoring, the online import of application data and online updating of application programs and UTM revision levels guarantee the uninterrupted high availability of UTM cluster applications. If a node application is terminated normally then users can continue open services at any other node application.
- For communications between a UTM application and a UTM cluster application, openUTM also provides automated load distribution via LPAP bundles in the case of LU6.1 and OSI TP communication.

- For communications between clients and a UTM cluster application, it is possible to distribute the load to the individual node applications by means of an external load distributor. In the case of UPIC communications, openUTM provides a UPIC load distributor for UPIC clients.
- Unlike standalone UTM applications, UTM cluster applications permit optimum load distribution with Oracle® RAC.

For more detailed information on the high-availability functions and load distribution in UTM cluster applications, see chapter “High availability and load distribution with UTM cluster applications”.

Administering a UTM cluster application

You can administer UTM cluster applications in the following ways:

- via separate administration programs that you create using the administration program interface (KDCADMI),
- via the administration command interface,
- via the WinAdmin and WebAdmin graphical administration tools (see "Simple, user-friendly application programming" and "Graphical administration with WebAdmin") which also use the administration program interface.

Depending on the administration task, any changes made either only apply locally in the individual node application to which you are logged on or globally in all node applications.

For more detailed information on administering a UTM cluster application via the program interface, see the openUTM manual “Administering Applications”.

Administering a UTM cluster application via WinAdmin and WebAdmin

WinAdmin and WebAdmin provide you with easy-to-use functions for the administration of UTM cluster applications. Both tools provide both a view that is global to the cluster and a view that is local to the node. WinAdmin permits administrative changes which apply globally to the cluster to be made only within the global cluster context, while administrative changes which apply locally to the node are possible only in the local node context.

WinAdmin and WebAdmin collect the data of all active node applications, collate it and display it in the user interface.

Alongside administrative changes to the node applications, you can also display the statistics data for all the node applications in the UTM cluster application.

It is also very easy to find out about the availability of the node applications.

For more detailed information on administering a UTM cluster application using WinAdmin and WebAdmin, see the associated online help system.

2.4.1 UTM cluster files

The configuration of a UTM cluster application comprises the following UTM cluster files which are accessed jointly by all the node applications:

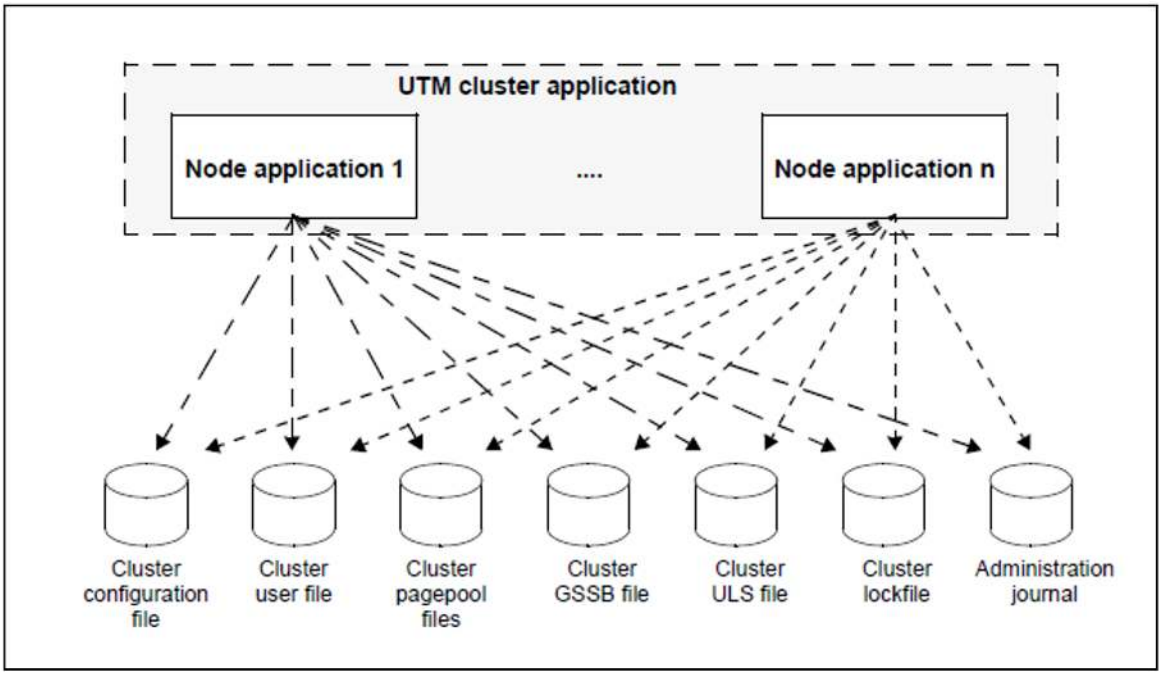


Figure 4: Shared synchronized access to the UTM cluster files from the node applications

The cluster configuration file, the cluster user file, the cluster page pool files and the cluster GSSB file and cluster ULS file are generated in a basic generation run on the initial generation of a UTM cluster application. The cluster lock file and the files of the administration journal are created by the application on initial start-up.

All the node applications must be able to access the UTM cluster files. To make this shared access possible, these files must be stored in the Network File System/Service (NFS) on Unix and Linux systems or on network drives on Windows systems.

Locking mechanisms across the different participating machines synchronize the access to these files.

The global cluster files are stored in a shared folder or with identical filename prefixes. They comprise the following files:

- For (UTM internal) cluster administration, there is the central **cluster configuration file** which contains the configuration of the UTM cluster application.
- The **cluster user file** is used to administer the users of a UTM cluster application. This file can be modified while an application is running if the administrator defines new users for the UTM cluster application or if a new KDCDEF run is performed for the application.
- The application data of a UTM cluster application that is valid throughout the cluster is stored in **cluster page pool files**. This includes, for example, the contents of the GSSB and ULS or user service data and therefore enables users to continue open services at another node
- The global memory areas GSSB and ULS are managed in a UTM cluster application via the **cluster GSSB file** and the **cluster ULS file**.
- The **cluster lock file** is used to manage locks for user data.

- These actions are written to the **administration journal** in order to permit changes that are global to the cluster to be implemented in all the node applications. All node applications reconstruct these changes on the basis of the administration journal.

  Actions with a global effect apply to all the node applications in the UTM cluster irrespective of whether they are currently active or not. Examples of such changes include the dynamic generation of objects such as new users (USER) or the modification of an object status. Administrators can initiate global actions from any node application in the UTM cluster application.

Alongside the **global** cluster files which are used by all node applications, **local** files also exist for each node, e.g. KDCFILEs (see "The KDCFILE - the “application memory”"), log files and diagnostic files. The KDCFILE is only generated once and is then copied once for each node application. At runtime, each node application uses its own copy exclusively.

The KDCFILEs for the individual node applications have to be set up in such a way that they can be accessed by all the node applications. This is necessary for application monitoring as well as for the online import of application data.

For more detailed information on the administration journal and on the file structure of a UTM cluster application, see the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems".

2.4.2 System requirements for the use of UTM cluster applications

The following system requirements must be satisfied before UTM cluster applications can be used:

- A Network File System/Service (NFS) must be available on Unix and Linux systems. This does not require the installation of any additional software on the node computer.
- There are no additional software requirements in the case of Windows systems. Shared files are accessed via the usual Windows network drives.

Refer to the Release Notice for more information on the listed software requirements.

- A cluster’s nodes must all run on similar operating systemsand in the same Bit mode, i.e. mixed configurations, such as Windows and Unix computers in combination, are not possible.
- The node computers of a cluster must all have the same "system platform class". The following "system platform classes" are supported:
  - Linux Systems
  - UNIX Systems
  - Windows systems

For more detailed information, refer to the Release Notice.

It is, in principle, possible to use different operating system versions at the individual nodes. In this case, please refer to the details regarding binary compatibility between versions. In most cases, upwards compatibility is possible.

For technical reasons, mixed system platform classes (e.g. Linux and Solaris) are not possible.

If the application works with databases, software that permits cross-machine access to the shared database must be installed on the system.

2.5 Message queuing

Message queuing (MQ) is a form of communication in which messages are not exchanged immediately, rather are buffered in intermediate queues before being dispatched. openUTM offers sophisticated message queuing functions through the concept of **asynchronous processing**.

The term “asynchronous” refers to a type of programming in which a program sends a message but need not wait for a response from the receiver program (non-blocking conversations). (non-blocking conversations). openUTM supports this programming style, of course, but it also offers a wide and subtly differentiated range of control options. Depending on who is responsible for processing the messages in a queue, a distinction is drawn between UTM-controlled queues and service-controlled queues.

UTM-controlled queues

A UTM-controlled queue is a message queue in which the retrieval and further processing of the messages is controlled entirely by openUTM. UTM-controlled queues are used for background jobs and output jobs. All MQ functions are available to both the sender and the recipient, which is why you do not have to generate or configure such queues and their queue managers separately or create any special triggering or polling mechanisms.

Service-controlled queues

A service-controlled queue is a message queue in which the retrieval and further processing of the messages is controlled by services. In other words, the recipient of the message is responsible for reading and processing it. The following types of service-controlled queue exist: USER queues, TAC queues and temporary queues:

- A USER queue is a user-specific message queue. A separate USER queue is automatically available to every UTM user. You can use USER queues to implement mailbox mechanisms for UTM users, for example.
- A TAC queue must be generated explicitly by means of KDCDEF (exception: dead letter queue). TAC queues can be addressed using their generated names by any service. TAC queues are used, for example, to forward UTM messages to the WinAdmin graphical administration workstation or to the Web application WebAdmin and archive them there.
- A temporary queue is created dynamically by a program and can also be deleted by a program. With temporary queues it is possible, for example, to implement a dialog between two independent services.

Deferred delivery mechanism with transaction management

openUTM works with the transaction-based deferred delivery mechanism in all message queues: the sender and recipient can run at different times and in different places, and the transmission of the message is guaranteed, regardless of whether or not there is a network connection. Messages are entered in a queue and stored there until the line and the recipient are ready. openUTM also offers additional flexibility in that time control and priority scheduling can be used in message queuing, and maximum wait times can be defined for service-controlled queues.

Dialog processing and message queuing can be combined as desired. MQ jobs can be initiated within a dialog service, and a service program started asynchronously can engage in a synchronous conversation with a remote dialog service. Long-running or non-time-critical jobs (e.g. slow print jobs, lengthy statistical calculations, sort operations, etc.) can be performed independently of the online dialog, without having to sacrifice the transaction processing capability.

Implementing modern concepts

Thanks to its message queuing functionality, openUTM is ideally suited to modern concepts, such as **workflow strategies**: operational procedures are divided into steps and intermediate states are passed from one application to the next. The sender program does not necessarily wait for a response from the receiver, and the next step may not be started immediately. However, it must be ensured that the intermediate state is actually received by the target program. This is the responsibility of the UTM transaction-oriented queuing mechanism. Even if the network is currently unavailable or the receiver application is offline, openUTM guarantees that no message will be lost or duplicated.

This independence from the quality and availability of connections means that openUTM forms a reliable middleware base for **mobile computing**. Mobile clients (e.g. applications) running on laptops can interact with servers without a permanent physical connection. Blocked transmission is made possible through the local, transaction-oriented collection of messages. Connection times are thus reduced and line costs kept to a minimum.

By using USER queues and TAC queues you can develop finely tuned concepts for **mailbox systems** and **alarm mechanisms** that include the clients, as well. This makes things much easier for the users of the application – both for customers and administrators – and thus means that the application is better received by users and that the security of the investment for the future is improved.

openUTM's message queuing functions also offer effective support for the implementation of **data warehousing solutions** and **decision support systems**. These applications generally work with vast, heterogeneous information pools, and involve linking and analyzing in-house and often external databases from the most varied IT systems. The information must be accessible to a large number of users. Since message queuing allows you to separate particularly long-running jobs from dialogs, openUTM ensures fast response times. In the case of pure retrieval applications in which restarting does not play such an important role, it is possible to use the UTM-F (**F**ast) variant (see section “Restart with UTM-F”).

Further information on message queuing can be found in chapter “Message queuing”.

2.6 openUTM - open for different platforms and protocols

Openness is one of the core principles of openUTM. It is reflected in numerous properties and functions, the most important of which are described below.

Cross-platform availability

These days, multivendor configurations are becoming the norm. openUTM is available

- for Linux distributions such as SUSE or Red HAT for example,
- for all conventional Windows platforms,
- for the conventional Unix platforms on request - such as Solaris, HP-UX or AIX,
- for BS2000 systems, for Business Server with /390 or x86 architecture.

Even the front-end openUTM-Client component is available for these platforms.

On other client platforms, such as MAC OS, client programs can communicate directly with UTM applications via a transport interface.

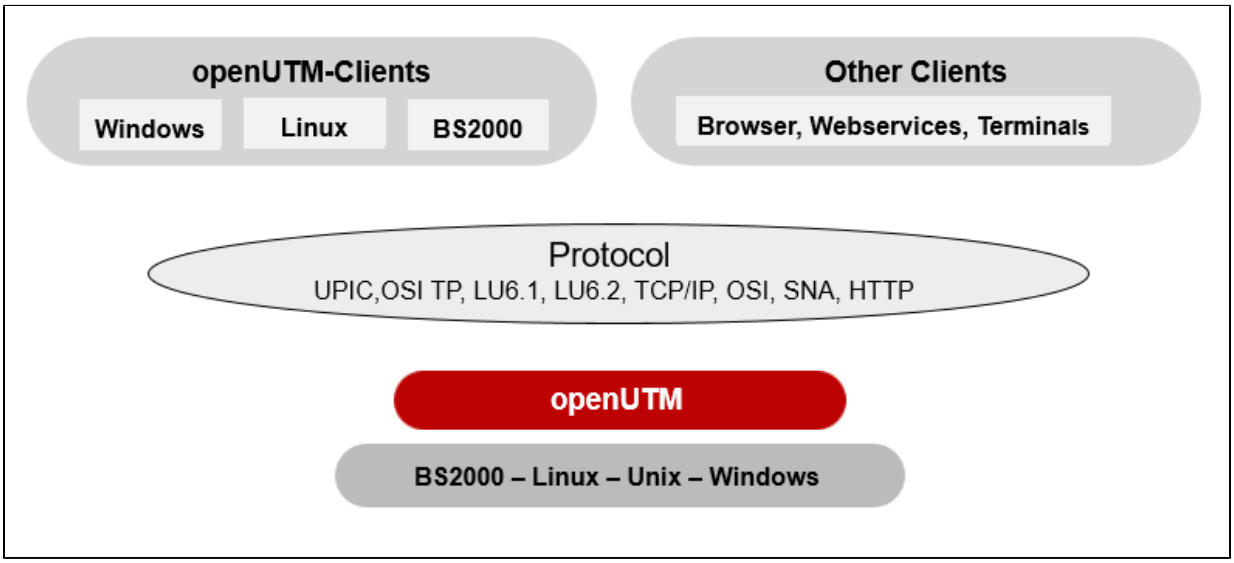


Figure 6: openUTM - available on a wide range of platforms

Due to the high portability of openUTM, the product can also be supplied for other Unix platforms (on request).

Because openUTM runs both on mainframes with the BS2000 operating system and on Unix, Linux and Windows systems, existing mainframe applications can be linked easily with new applications on Unix and Linux systems or Windows systems. Small departmental servers can thus be integrated quickly and flexibly in a system of existing mainframe applications, for example.

Integration into IBM mainframe environments

openUTM supports the LU6.1 and LU6.2 communication protocols. UTM applications can therefore interact with CICS-IMS/TM or IMS/DC applications while using transaction management. See section “Communication with CICS, IMS and TXSeries applications” for more information.

APPC/CPI-C applications in IBM environments can also be connected to UTM applications. CICS applications (without transaction management) can also be linked directly via TCP/IP.

openUTM is thus ideal for all downsizing, rightsizing or connectivity projects in IBM mainframe environments - both from a technical and an economic point of view.

Connecting to other transaction monitors via OSI TP and LU6.2

Based on the standardized, open communication protocol OSI TP, it is possible to interact directly with all transaction monitors that support OSI TP. Thus, for example, openUTM supports direct, transaction-oriented interaction with Tuxedo applications or with applications in UNISYS environments.

Connecting terminals

In addition to offering interfaces for connecting client programs, openUTM also allows terminals to be connected. These include, for example:

- older devices, such as alphanumeric terminals
- programs that emulate such terminals (e.g. 9750 emulations)
- network terminals controlled by PC software that does not permit all the functions of a PC to be used

These terminals can be used in line mode. On BS2000 systems you can use screen formats (masks) when openUTM is operating together with formatting systems such as FHS. By means of products such as WebTransactions, screen forms can be converted dynamically into graphical user interfaces and thus integrated into Microsoft Office environments or Web applications, for example.

openUTM provides special user commands for direct communication with terminal users.

The user commands are described in the openUTM manual “Using UTM Applications”. Information on how to use the terminal interface for your service routines can be found in the openUTM manual “Programming Applications with KDCS for COBOL, C, and C++”. Formatting tools are described briefly in the present manual, see section “Formatting” for BS2000 systems.

Integration in the World Wide Web

A UTM application can act as an HTTP server and communicate with HTTP clients. This allows a UTM application to offer its services to the outside world and any HTTP clients, such as browsers or e.g. cURL clients, can access these services.

An overview on the HTTP functionality of openUTM can be found in chapter "Communicating with HTTP Clients"

openUTM can as well be integrated into the World Wide Web using the "WebServices for openUTM" product, the WebTransactions product or the openUTM-JConnect product delivered with the BeanConnect product. UTM applications can then be reached world-wide from millions of computers using Web browsers such as Firefox or Internet Explorer through a uniform and modern graphical interface.

You will find additional details on section “Addressing openUTM via Webservices” and section “Java clients”.

Interaction with any applications

Almost any conceivable type of device can be connected as clients to openUTM: sensors, control systems or industrial automatons, for example. The only requirement is that they provide an interface at the level of the transport system. In particular, applications can be connected to openUTM via the widely used **socket** interface. For socket interfaces openUTM supports the protocol HTTP as well as the openUTM-specific socket protocol (USP).

As a result, UTM services can be requested by all kinds of devices: via the transport interface, the client device simply passes the appropriate service name together with any data that may be needed openUTM (see also section “Communicating with transportsystem applications”).

Support for the most varied communication and network protocols

openUTM’s support for the most varied communication and network protocols forms the basis for the integration of heterogeneous environments.

openUTM supports different network protocols, e.g. TCP/IP via RFC1006 or TCP/IP native via the socket interface. and in WANs (wide area networks, e.g. X.25). Users and programmers need not concern themselves with the different networking technologies; your UTM applications will run in any environment.

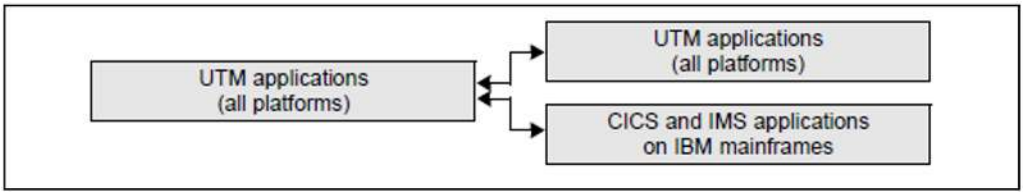
openUTM not only supports a wide range of network protocols, but also various high-level communication protocols. These protocols govern types of interaction that go well beyond the mere exchange of data. For example, they ensure that applications from different manufacturers can interact with each other on the basis of global transaction management.

The table on the next page provides an overview of the high-level communication protocols supported and the connection options offered by these protocols.

Overview: High-level communication protocols and connection options

*LU6.1 (**L**ogical **U**nit **6.1**)*

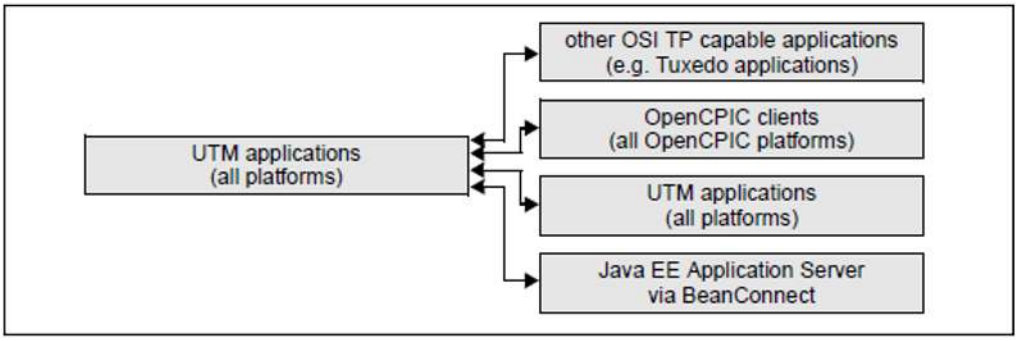
The LU6.1 protocol is an SNA protocol defined by IBM. Following ongoing development, it has now become an industry standard. Communication takes place on the basis of global transaction management. LU6.1 is particularly suitable for the following connection options:



*OSI TP (**O**pen **S**ystems **I**nterconnection **T**ransaction **P**rocessing)*

International standard for Distributed Transaction Processing defined by ISO. It is possible to define whether or not communication is to take place on the basis of global transaction management.

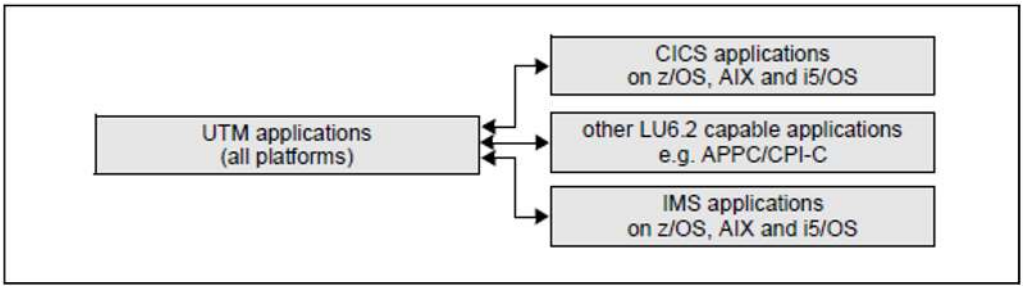
OSI TP is particularly suitable for the following connection options:



*LU6.2 (**L**ogical **U**nit **6.2**)*

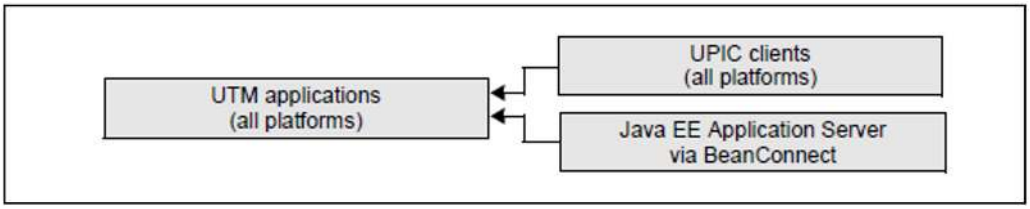
A protocol defined by IBM. This protocol is supported by openUTM via openUTM-LU62. openUTM-LU62 is an OSI TP partner from openUTM's point of view. In this manner, you can work with and without global transaction management.

openUTM-LU62 is suitable for the following connection options:



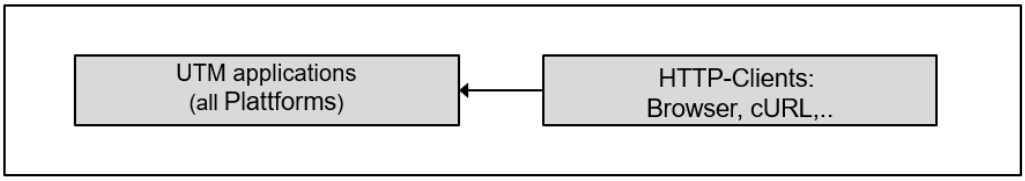
*UPIC (**U**niversal **P**rogramming **I**nterface for **C**ommunication)*

UTM interface and protocol for connecting front-end Clients



*HTTP (**H**yper**t**ext Transfer **P**rotocol)*

The Hypertext Transfer Protocol is a stateless protocol for transferring data on the application layer over a computer network. An openUTM application is able to communicate with HTTP clients. Communication can take place via HTTP or HTTPS connections. A more detailed explanation of these terms can be found, for example, in Wikipedia.



2.7 X/Open conformance of openUTM

The openness of openUTM is also reflected in its conformance with X/Open:
openUTM complies with the reference model for Distributed Transaction Processing (DTP) defined by X/Open, and supports the interfaces standardized by X/Open.

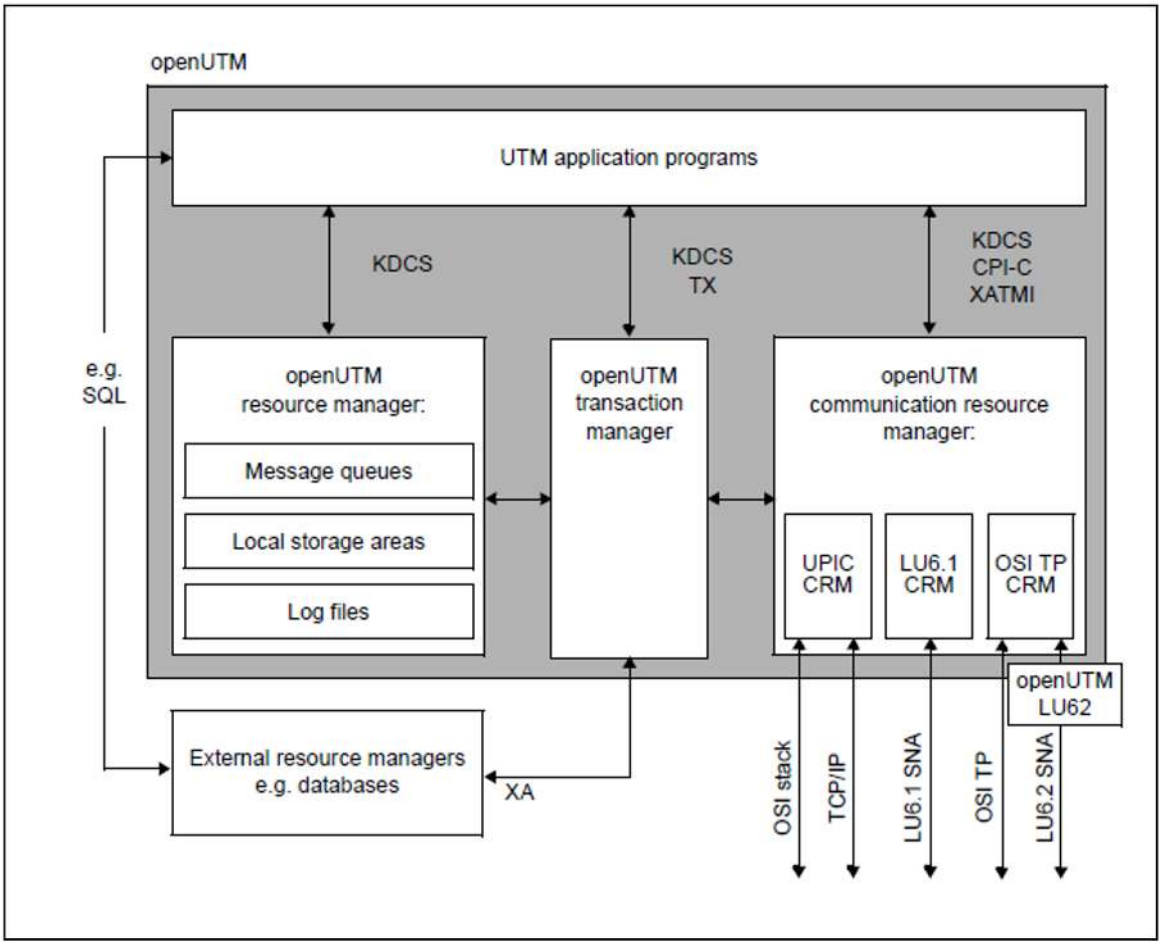


Figure 7: Architecture of openUTM as defined in the X/Open model

X/Open differentiates between four component types in transaction-oriented systems: application programs, transaction managers (TMs), resource managers (RMs), and communication resource managers (CRMs).

- Application programs:
  Application programs implement the processes to be performed, and access the services of the other components via standardized program interfaces.

  In addition to the CPI-C, XATMI, and TX program interfaces standardized by X/Open, openUTM supports the universal KDCS interface (German standard).
- Transaction managers (TMs):
  Transaction managers are responsible for controlling and monitoring transactions, and provide recovery functions. They coordinate access to data and communication resources. In the case of external resource managers, e.g. database systems, this takes place via the XA interface.

- Resource managers (RMs):
  Resource managers manage data resources. An example of an RM is a database system. openUTM also provides its own resource managers, e.g. for accessing message queues, local storage areas, and log files. Application programs access the RMs via RM-specific interfaces, usually SQL for database systems and the KDCS interface for openUTM RMs.
- Communication resource managers (CRMs):
  Communication resource managers control communication between the application programs in distributed systems.
  openUTM provides CRMs for the international OSI TP standard, the LU6.1 and LU6.2 industry standards, and the openUTM-specific protocol UPIC (see also section“openUTM - open for different platforms and protocols” and sections “Communication with UTM-Client applications” through “Overview: partners, protocols, transaction management”). In openUTM, the OSI TP connection is implemented via the XAP-TP System Programming Interface standardized by X/Open.
  The applications programmer can access the communication options of the CRMs via the CPI-C, XATMI or KDCS program interfaces.

The appendix includes a list of all standards supported by openUTM.

Advantages of X/Open conformance

The X/Open conformance of openUTM offers a range of advantages:

- option of porting application programs through standardized program interfaces (e.g. CPIC)
- option of integrating heterogeneous distributed systems through standardized communication protocols (e.g. OSI TP)
- option of replacing components, e.g. RMs, through standardized integration interfaces (e.g. XA)

Since openUTM can also be used on all conventional hardware platforms and has excellent connectivity properties (see section “openUTM - open for different platforms and protocols”), you can distribute your application in a heterogeneous environment in accordance with the procedures used in your company. Existing application program units - even those under other transaction monitors - can be integrated without any problems, thereby protecting your investment.

The openUTM transaction monitor thus provides you with all the options required to define an application architecture appropriate for your business procedures in a heterogeneous IT environment, and to implement this architecture with the help of suitable interfaces.

2.8 Performance, throughput and response times

One of the main strengths of openUTM is its high performance levels.
openUTM aims to provide the highest throughput rates and fastest response times of any system currently available. This level of efficiency is achieved by means of a finely tuned system resource management facility, e.g. through the use of multithreading techniques and through automatic, dynamic load management. Ideally, you should use multiprocessor hardware.

Time-consuming jobs can be run separately from online processing with the help of the message queuing functionality.

In the case of online dialogs, openUTM optimizes wait times, such as those that give the terminal user time to think, by allocating the resources to other jobs. This eliminates the problem of blocked processes. openUTM can thus distribute a large number of simultaneous requests across a small number of processes.

openUTM also provides a sophisticated priority scheduling concept that can be used for dialog processing as well as for background jobs (see "Priority scheduling of background jobs").

While in conventional solutions the system overhead and thus the response time at least increase proportionally with the number of users, the response time of openUTM remains well below these values. This has been verified by comparative measurements, even with relatively low numbers of users.

openUTM is equally suitable for large configurations with thousands of clients working in parallel and transaction rates of several million transactions per day.

Performance control with KDCMON

If performance bottlenecks arise, you can carry out an extensive analysis with the UTM monitor **KDCMON**. KDCMON records numerous pieces of detailed information on the operation of UTM applications and program units. KDCMON returns information on wait times or on the resource utilization of individual services, for example.

You can activate KDCMON while the system is running and deactivate it after the desired measurement period has passed. Data can be recorded from one or more UTM applications on a computer. The measurement data is evaluated using the KDCEVAL program and can then be transferred to a PC and displayed there in the form of a chart.

The KDCMON monitor is described in detail in the respective openUTM manual “Using UTM Applications”.

2.9 Workload Capture & Replay

The simulation of load situations under real conditions is an important capability that makes it possible to test and optimize modules and improve performance.

"Workload Capture & Replay" provides functions which allow you, for example, to compare identical UTM applications from different openUTM versions or perform comparative measurement on different computer hardware variants. The Replay capability even makes it possible to compare different system platforms. Of particular importance is the ability to control the simulation of predicted higher future loads by simply adjusting the scaling parameters during replay.

Workload Capture & Replay is available for application loads that were generated by UPIC clients without encryption.

Process steps during Workload Capture & Replay

Workload Capture & Replay is performed in the following steps:

1. UPIC Capture
   Communication between the UTM application and the UPIC clients is recorded during live application operation. To do this, UTM uses trace functions and the recording is available in the form of trace files that are local to the process.
2. Trace merging
   The recording of all the UTM processes is now sorted and entered in a file in the precise chronological order.
3. UPIC Analyzer
   The sorted recording is analyzed using the program **UpicAnalyzer**. This results in the generation of a file in UPIC Replay Format.
4. UPIC Replay
   The file in UPIC Replay Format is used as the input for the UPIC-based load driver program **UpicReplay**. During the load test, the recorded UPIC conversations are automatically repeated in the correct chronological order. To simulate different loads, scaling parameters can be adjusted to vary the number of UPIC clients that are active in parallel and the thinking time per client.

   For the replay operation, the load, and therefore also the throughput, can be set to be several times greater than during recording. This replay operation is reproducible and can be repeated multiple times in order, for example, to examine the behavior of the UTM application at different load factors.

The programs **UpicAnalyzer** and **UpicReplay** run on 64-bit Linux systems.

2.10 High availability

Downtimes are simply not acceptable for applications used in the mission-critical areas of a company. openUTM ensures that your applications are available around the clock (7*24h-hour operation):

- UTM applications can be configured dynamically, i.e. maintenance activities such as modifying and extending the configuration (e.g. when connecting additional clients) can be carried out online.
- It is possible to replace all or parts of the application program “on-the-fly”, i.e. without affecting the availability of the system at any point.
- All log files can be switched over during live operation so you can then evaluate and save them or delete them.
- In the event of errors in a service program, the effect remains local. At worst, the process is terminated (and then automatically replaced) without affecting other services or the application as a whole. There is therefore no single point of failure. If desired, it is possible to configure the event-driven initiation of an alternative service.
- Even hardware errors in peripherals, e.g. printers or terminals, do not jeopardize availability, since openUTM offers switchover functions which are activated automatically or with the help of an administration command.
- If an application crash is inevitable, e.g. because the server has crashed following a hardware error, the UTM application can be restarted immediately. The restart functions ensure that the interrupted services and dialogs continue from where they left off using consistent data.
- Because the “memory” of the UTM application, the KDCFILE, can be kept in duplicate on different disks, the operability of the application is not threatened even if one of the disks is physically destroyed.
- In conjunction with high-availability hardware and software, openUTM can be made to cope with even the most extreme requirements (see also "High availability with standalone UTM applications").
- On Linux, Unix and Windows systems openUTM offers greater availability thanks to its cluster support. A UTM application can run as a UTM cluster application on a cluster. This high availability of UTM cluster applications is guaranteed, in particular, by the possibility of modifying the configuration or application program or implementing a new UTM revision level while the UTM cluster application is running. This functionality is also referred to as "online updating". This functionality is complemented by other central functions for high availability such as application monitoring and the online import of application data (see chapter “High availability and load distribution with UTM cluster applications”).

You will find more information on the subject of availability in the following manuals: the openUTM manual “Generating Applications” (on keeping the KDCFILE in duplicate), the openUTM manual “Using UTM Applications” (on changing programs) and the openUTM manual “Administering Applications” (on dynamic configuration).

2.11 Security functions

openUTM offers comprehensive, distinct, clearly structured concepts for system and data access control (authentication and authorization):

- Definition of logical access points:
  A UTM application can be configured in such a way that a client can only connect to it if a logical access point (LTERM partner) is configured for it in this application. In this case, the client must therefore be known to the UTM application.
- Definition of user IDs:
  User IDs can be defined for a UTM application. This can take place during generation, or dynamically while the application is running.
- Allocation of passwords to user IDs:
  Specific passwords can be assigned to the user IDs. You can also specify the complexity and the period of validity of the passwords and keep a history of the passwords used.
- Use of an ID card reader for system access control
- Silent alarm in the case of repeated failed attempts to log on by a user.
- Automatic timeout
- Event-driven routines for user-defined system access controls
- Subtly differentiated access authorizations:
  openUTM offers two different concepts, which each take a different angle:
  - The lock code/key code concept, which is user-oriented
  - The access list concept, which is role-oriented

  The two concepts can be used in parallel within an application.
- Password encryption and encryption of messages on the way between the client and server.
- A cross-application user concept in cases where OSI TP is used.
- Support for Kerberos on BS2000 systems (for the terminal mode)
- Definition of IP subnets:
  Restriction of the IP address range by defining IP subnets (SUBNET) for client access via LTERM pools (TPOOL).

You will find detailed information on the subject of security in chapter “Security functions”.

The extensive system and data access control concepts offered by openUTM are also available when connecting UTM-Client programs and during distributed processing with other partner applications.

When coordinating with databases, the protection mechanisms of the database systems can also be used. These mechanisms are described in the documentation for the relevant database systems.

Further information on the security functions can be found under the keywords ACCESS-LIST, LTERM, KEYSET, KEYS, and LOCK in the openUTM Manuals “Generating Applications” and “Administering Applications”.

2.12 Dynamic configuration

Demands are being made of OLTP applications to the effect that it should be possible to change configuration data dynamically in order, for example, to add new users or services to an application, without interrupting the running of the application.

In a UTM application, configuration data such as information on user IDs, communication partners, buffer sizes, etc. is initially specified statically when the application is generated. openUTM offers powerful tools and interfaces that allow you to change the configuration data while the application is running.

The graphical administration program WinAdmin offers a complete, Java-based solution for dynamic configuration. WinAdmin allows you to change the generated application parameters or add or delete generation objects from any Java platform (e.g. user IDs or connections to partners). You will find more information on WinAdmin on "Simple, user-friendly application programming" and "WinAdmin graphical administration program".

The WebAdmin component provides a web-based user interface for the administration of UTM applications on all platforms. The scope of functions provided by WebAdmin is similar to that found in WinAdmin. For further information on WebAdmin, see "Graphical administration with WebAdmin" and "WebAdmin graphical administration program".

The KDCADMI program interface allows you to create programs for dynamic administration that are designed to match your own requirements as closely as possible, in order, for example, to schedule specific changes and have them run automatically. You will find more information on this on "Administration program interface".

The UTM tool KDCUPD allows you, after you have regenerated your UTM application, to transfer user data and administration information of the application to the new configuration. KDCUPD is used when fundamental changes have to be made to the generation or when changing to a different version of UTM. When KDCUPD is used the following applies:

- The running of a standalone application is briefly interrupted. After this, all the users can resume their work at the position at which they were previously interrupted.
- In the case of most changes, UTM cluster applications can continue to run without interruptions since the individual nodes only have to be interrupted one after the other.

You will find more information on "Updating the configuration using the KDCUPD tool".

2.13 Internationalization/adaptation of UTM messages

openUTM provides a wide range of internationalization functions for creating multilingual UTM applications, which are then made available to users in their own language. Dates, times, units of measurement, and currency symbols are displayed in accordance with local conventions.

There are a number of easy-to-use options for adapting the UTM system messages, which are supplied in English and German as standard. These options allow you to incorporate messages in other languages, replace or modify standard texts, and redefine other message properties, such as the output lines.

A UTM application can work with several message files, which means that each user can be supplied with individually tailored messages. This provides a great deal of flexibility when designing applications.

openUTM for Unix and Linux systems complies with the internationalization guidelines defined in the X/Open Portability Guides.

You will find more information on internationalization on BS2000 systems in section "BS2000-specific functions".

Detailed information on message formats (e.g. the various message lines) and the adaptation options available can be found in the openUTM manual “Messages, Debugging and Diagnostics”.

2.14 openUTM in the Unicode environment

A UTM application can be converted without a problem to a Unicode environment and can be operated in a Unicode environment. For example, openUTM can work with databases whose records are available in the Unicode format and can exchange Unicode data with other applications. openUTM operates transparently when using Unicode data.

If you want to convert application programs to process Unicode data, then the corresponding compiler must be capable of handling Unicode.

The openUTM administrative data such as user IDs, transaction codes, etc., are also defined in a Unicode environment in a 7-bit code.

2.15 Accounting

openUTM provides functions that allow computer centers to charge users of a UTM application for the computing power used. Users are understood to be the UTM user IDs with which a user signs on. In the case of distributed processing, the session (LU6.1) or the association (OSI TP without cross-application user concept) is used instead of the user ID, so that accounting is also possible under these circumstances.

Accounting can be based on actual usage (consumption) or using a fixed price.

Fixed price

When fixed-price accounting is used, openUTM charges the user account a specific number of accounting units every time a specific service is called. In order to determine the fixed price of a given service, openUTM can record the resource utilization of services, such as the average CPU consumption. Using these records, the number of units to be charged for the individual services can be determined via generation. Some services may be free, such as informational services.

Consumption accounting

When consumption accounting is used, openUTM determines the current resource utilization of a user (according to seconds of CPU time, for example) and charges the units used to the user’s account at predefined intervals. Depending on the system, the accounting process can distinguish various resources, such as the CPU, printer output or input/output, and weight their usage differently.

Accounting can be activated or deactivated while the system is running or separately via generation.

Accounting with openUTM on the various platforms is described in detail in the respective openUTM manual “Using UTM Applications”.

2.16 Performance monitoring with the openSM2 Software Monitor

The **SM2 software monitor** supplies statistical data on the performance and capacity of your system's resources. openSM2 allows you to monitor the behavior of a UTM application during live operation and to locate any performance bottlenecks. You can view the data supplied by openUTM online on an openSM2 screen. In addition, openSM2 also stores the data in a file for later evaluation. The evaluation of monitored data reveals the behavior of the entire application, e.g. mean values for transaction rates, throughput and processing times.

The openSM2 software monitor is described in a separate manual entitled "openSM2 Software Monitor". For details on the interaction between openSM2 and openUTM, see the openUTM manual "Using UTM Applications on BS2000 Systems" and openUTM manual "Using UTM Applications on Unix, Linux and Windows Systems" respectively.

2.17 Diagnostic capabilities in openUTM

openUTM offers you comprehensive support when diagnosing and locating errors in the application.

Errors in function calls are reported by openUTM using appropriate **return codes** and messages. If you wish, you can request a **dump** in order to analyze the error situation more closely. This request is issued either event-driven or using an administration command or a call in the program. If an error causes an application process or the UTM application to crash, a process-specific dump is automatically generated.

The dump file contains all KDCS and administration calls as well as database calls made by openUTM. On BS2000 systems the dump file also contains calls to the formatting system and database calls made by the program (when using IUTMDB). This makes it easier to localize errors when working with databases, for example.

To analyze the dump file, openUTM provides a special tool (KDCDUMP) which converts the dump file to a printable format. This tool also allows you to evaluate the dump file directly from the terminal, e.g. to search for particular table entries and format these entries. Of course, you can format the entire file, and then print it.

openUTM also offers **trace functions** for monitoring events. For instance, the trace functions can be used to log all connection-related activities within a UTM application or all activities relating to OSI TP connections in a trace file. To analyze the trace files, you can use the special tools provided by openUTM. The trace functions can be enabled when the application is started using an administration command or via the administration program interface.

The application-specific systemlog file **SYSLOG** (SYStem LOGging) also contains important diagnostic information, and is created by openUTM when an application is started. The SYSLOG file contains messages, which may be useful for ongoing monitoring or subsequent analyses. You yourself can determine the selection of messages to be logged by assigning the SYSLOG message line to certain messages. UTM tools are available for formatting and analyzing the SYSLOG file.

A detailed description of the diagnostic options available can be found in the corresponding openUTM manual ”Messages, Debugging and Diagnostics”, which also contains information on the KDCDUMP dump analysis tool and on the tools provided for analyzing the SYSLOG system log file.

2.18 Simple, user-friendly application programming

To program the service routines, you can use the usual programming languages: the UTM calls can be integrated into normal C, C++ or COBOL programs. On BS2000 systems, the KDCS interface is also available for Assembler, Fortran, PASCAL-XT and PL/I. Even if the service routines are encoded in different programming languages, they can be combined as desired.

When programming applications, you are not restricted to any particular communication model, since openUTM supports a wide range of different models (conversations, pseudo-conversations, request/reply, message queuing).

The UTM HTTP interface is only available for C and C++ programs.

Programming is also simplified considerably due to the fact that many central tasks are performed automatically by openUTM.

For instance, openUTM handles:

- control of global transactions
- management of parallel accesses
- establishment of network connections
- precautions for error situations.

You can therefore program your routines as if creating an application for a single user in a standalone, homogeneous system.

Ideal development environments

When creating a UTM application, you can make full use of all the resources provided by the operating system and the compilers.

NetCOBOL from Fujitsu and VisualCobol from Micro Focus are the development environments for COBOL users on open platforms.

C and C++ users are supported effectively by Oracle Studio from Oracle Corporation and the Microsoft Visual Studio.

BS2IDE - Eclipse-based integrated development environment for BS2000 systems

You can also use BS2IDE to create UTM applications on BS2000 systems. BS2IDE is available as a plug-in to the Eclipse open development environment. This plug-in supports developers during typical tasks and incorporates the advantages of modern Integrated Development Environments (IDE).

BS2IDE combines the most important tools from the software development process in a user interface and assists developers during troubleshooting.

2.19 Graphical administration with WinAdmin

From Windows, Unix or Linux systems, you can administer UTM applications running on all platforms using a comfortable graphical interface with the WinAdmin openUTM component:

- Since WinAdmin supports the complete scope of the KDCADMI administration program interface, one or more UTM applications and one or more UTM cluster applications can be administered and configured dynamically, i. e. by adding new objects or deleting old objects.
- For the administration of UTM cluster applications, WinAdmin provides you with administration functions which only apply to node applications as well as administration functions that are global to the cluster which apply to all the nodes in the UTM cluster application. You will find more information on "WinAdmin graphical administration program".
- Alongside the administration program interface, WinAdmin also provides the classical command interface for applications. You will find more information on "WinAdmin graphical administration program".
- In addition, WinAdmin is able to generate statistics and display them both in table form and in form of charts.

The UTM applications may be distributed arbitrarily in the network on different platforms. The UTM applications to be administered can be grouped together in collections so that they can be administered together. For example, it is possible to modify objects in **several** applications in **one** step.

openUTM WinAdmin communicates with the UTM application as a UPIC client and runs under all conventional Unix, Linux and Windows systems.

Security in WinAdmin

Of course, you are provided with all UTM security functions for administration via WinAdmin, from system access protection to UTM user IDs and passwords through the encryption of passwords and data.

WinAdmin also offers its own user concept because extensive security demands are often placed on administrators:

- You can define several users and assign them different authorizations, from users with read access only right up to the “master” user, the WinAdmin administrator. The access rights can also be restricted to specific applications or object types.
- Access to WinAdmin can be protected by a password for every user.

The WinAdmin documentation (online help and WinAdmin description available online in the form of a PDF file) tells you about the other features of openUTM WinAdmin and how to use the product.

2.20 Graphical administration with WebAdmin

With the WebAdmin component, openUTM provides you with a web-based user interface for the administration of UTM applications on all platforms. WebAdmin runs on a web server and can be called via a browser running on any client.

- It permits the administration and dynamic configuration of UTM applications and UTM cluster applications. This means that you can both add and delete objects because WebAdmin supports the full function scope of the KDCADMI administration program interface.
- When administering UTM cluster applications, WebAdmin provides you both with administration functions that apply only to a single node application as well as with global, cluster-level administration functions that apply to all the node applications in the UTM cluster application. For more detailed information, see "WebAdmin graphical administration program".
- Alongside the administration program interface, WebAdmin also provides the classical command interface for applications. You will find more information on "WebAdmin graphical administration program".
- In addition, WebAdmin is able to generate statistics and display them both in table form and in form of charts.

The UTM applications that are to be administered can distributed across different platforms anywhere in the network. They can be grouped together to form collections.

openUTM WebAdmin is interfaced to the UTM application as a UPIC client and runs on all commonly used Unix, Linux and Windows systems. It also runs as an add-on on the management unit of an SE server.

Security in WebAdmin

When performing administration tasks in WebAdmin, you naturally benefit from all the UTM security functions, from access control through UTM user IDs and passwords and on to password and data encryption.

However, because administrators are faced with particularly exacting security demands, WinAdmin also provides a separate user concept:

- You can define multiple users and assign them different rights, starting from users with simple read-only rights through to the "master" or WebAdmin administrator.
- Access to WebAdmin can be password-protected for each user individually.

You can find more information about the performance features of openUTM WebAdmin and the ways of using this product in the WebAdmin documentation (online help and WebAdmin description which is available online as a PDF file).

2.21 SNMP subagent for openUTM

The SNMP (Simple Network Management Protocol) management protocol belongs to the TCP/IP protocol family and is the de-facto standard for system and application management in addition to being the standard for network management.

The SNMP subagent for openUTM binds the openUTM transaction monitor in a distributed control center and

- allows you to obtain a complete overview of all objects belonging to a UTM application such as system parameters, physical and logical terminals, TACs, users, etc., for selected UTM applications,
- integrates UTM applications in the graphical network card of an SNMP manager and allows you to display the states in color and
- provides administration functions such as the ability to change the properties of an application, locking and unlocking clients, starting and terminating an application.

The subagent communicates with the UTM application being monitored via the CPI-C interface with UPIC as carrier system and can only be connected to one application at a time.

The SNMP Master Agent with its SNMP subagents can be connected in principle to all management centers via SNMP, e.g. to the CA Unicenter. This product offers all capabilities for the integration of any system with private MIBs (Management Informations Bases).

3 Integration scenarios with openUTM

openUTM is used more and more often with other applications, in particular with Java Enterprise Edition technology. When used with other applications, the Service Oriented Architecture (SOA) plays an important role. openUTM is easily integrated into these architectures, and in this case openUTM can act as a called service or take on the role of the calling “client”.

openUTM as part of openSEAS

openSEAS is a coordinated suite of products that meets the particular demands currently placed on IT systems (Web connection or integration into other systems, for example). openUTM is fundamental to the openSEAS concept.

3.1 Integrating different applications

The design of integrated business processes consisting of several applications is becoming increasingly central to the work of IT departments. There are two different scenarios to examine here:

- The UTM application represents the logic of the business process (or of a sub-process) and another application is to be called from the UTM application as a service. In this case, the UTM application can access a Java Enterprise Edition application via BeanConnect.
- The program units of the UTM application provide services for the business process applications.

These two scenarios are described in more detail in the following sections.

3.2 Integrating openUTM in the Java Enterprise environment

The Java Enterprise Edition technology (Java EE), defined by Oracle Microsystems, is becoming increasingly important when it comes to the creation of server applications. One reason for this is the universality of Java as a programming language, and the other is the component technology itself. This allows program segments to be put together like building blocks, regardless of whether they are components developed in-house or standard components widely available on the market. One example of this are e-commerce solutions, which consist partly of services developed in-house and partly of general services (shopping basket management system, for example).

openUTM is particularly suitable for use in environments with Java EE application servers because the **BeanConnect** product enables Java EE solutions to be optimally integrated into openUTM-based solutions.

BeanConnect is an adapter that is based on the Java Connector Architecture (JCA) of Oracle and that supports standardized connection of openUTM applications to Java EE application servers, in particular to Oracle's application server. Other application servers such as IBM's WebSphere can also be used as well, but in this case a separate service package needs to be purchased due to differences between the application servers.

3.2.1 openUTM as a server for Java EE application servers

When linked, openUTM can operate as a server. In this case, an application (EJB) on the Java EE application server starts communication by sending a message to the UTM application. This is outbound communication from the point of view of the Java EE server. The link can be established in one of two ways:

- As a pure client connection via the UPIC protocol
- As a server-server link via OSI TP

These two links are implemented using components of the BeanConnect product, see the following figure:

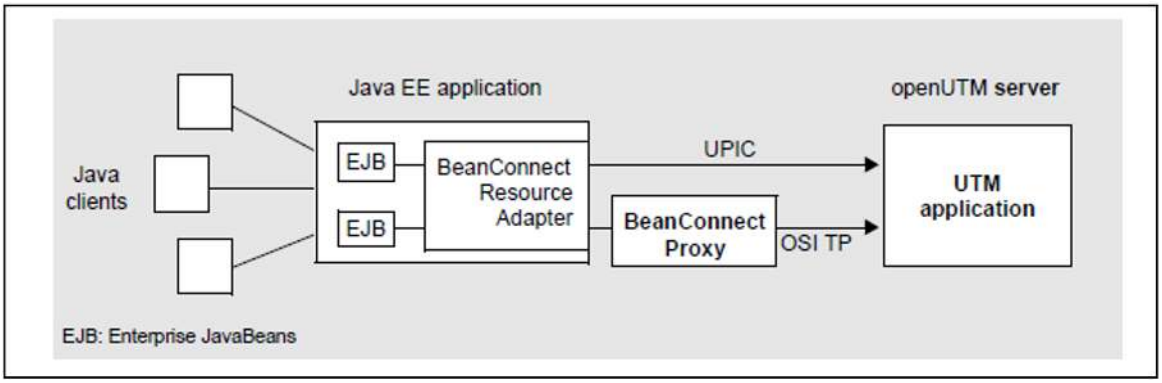


Figure 8: openUTM as a server for a Java EE application server

Linking via UPIC

From openUTM's point of view, this type of link corresponds to a classic client connection via the UPIC protocol, see section "Clients with the UPIC carrier system". In this case the Java EE application can call every openUTM dialog TAC, but cannot call any asynchronous TACs.

Due to the use of the UPIC protocol, the Java EE application is integrated in the restart concept of openUTM, i.e. the Java EE side can request the status of the last transaction, for example. However, the Java EE application must always initiate this; global transaction management is not possible.

This type of link is suitable for simple integration scenario such as mere requests for information or single-step transactions that do not require distributed transactions. This simple "entry-level" adapter is an attractive solution for these kinds of scenarios.

Linking via OSI TP

From openUTM's point of view, this type of link corresponds to a server-server link, see "Server-to-server communication". In this case the BeanConnect Proxy component acts as an intermediary. In contrast to connection via UPIC, linking via OSI TP offers you the additional capability of implementing sophisticated integration scenarios as well:

- The applications can work with or without distributed transaction management
- The Java EE application server can call dialog and asynchronous TACs in the UTM application

To establish a OSI TP link, you must modify the generation of the UTM application and configure the BeanConnect Proxy accordingly.

3.2.2 openUTM as a client for a Java EE application server

openUTM can act as a client of a Java EE application server when the UTM application initiates communication by sending a message to the J2EE application server. This is inbound communication from the point of view of the J2EE server. The OSI TP protocol or a transport system protocol can be used for communication, see the following figure:

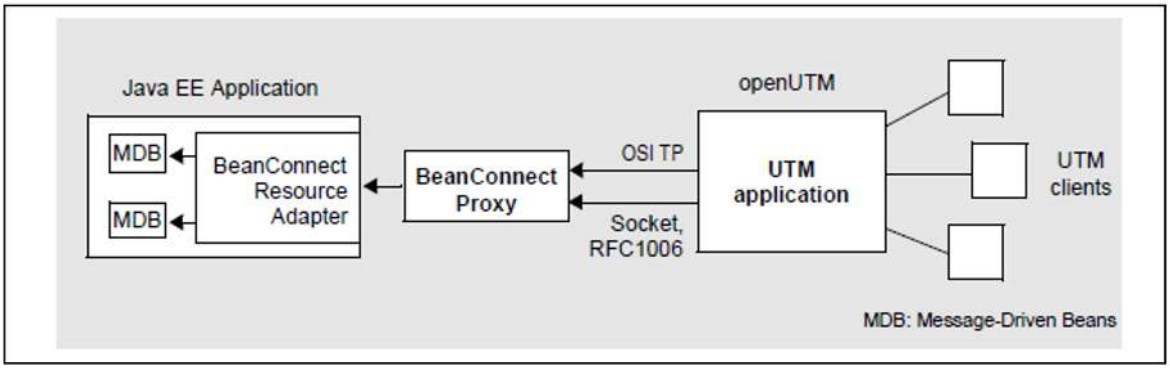


Figure 9: openUTM as a client of a Java EE application server

The link between openUTM and the Java EE application operates via the BeanConnect Proxy component. The messages sent by the UTM application are sent to message endpoint applications (message-driven Beans) on the Java EE application server.

Communication can take place via OSI TP or the transport system protocol:

- When the OSI TP protocol is used, you can work with or without transaction management. Asynchronous communication and communication in the dialog is possible.
- If a transport system protocol such as Socket or RFC1006 is used, then communication is always asynchronous and non-transactional.

To establish a link, you must modify the generation of the UTM application and configure the BeanConnect Proxy accordingly.

Use the BeanConnect Management Console to configure BeanConnect Proxy. For further information, refer to the manual "BeanConnect".

3.2.3 UTM cluster application as client or server

A UTM cluster application in a connection can function as a server outbound communication and as a client for for inbound communication, with an application (Enterprise Java Bean, EJB) in the Java EE application server starting communications by sending a message to the UTM cluster application. From the perspective of the Java EE server, this is outbound communication. In the case of outbound communication, it is possible to address the individual nodes of the UTM cluster application using the round robin method. The connection can be implemented either as a server-server link via OSI TP (see also section “Linking via OSI TP” (openUTM as a server for Java EE application servers)) or as a client link via the UPIC protocol.

This connection is implemented via the BeanConnect product, see the figure below:

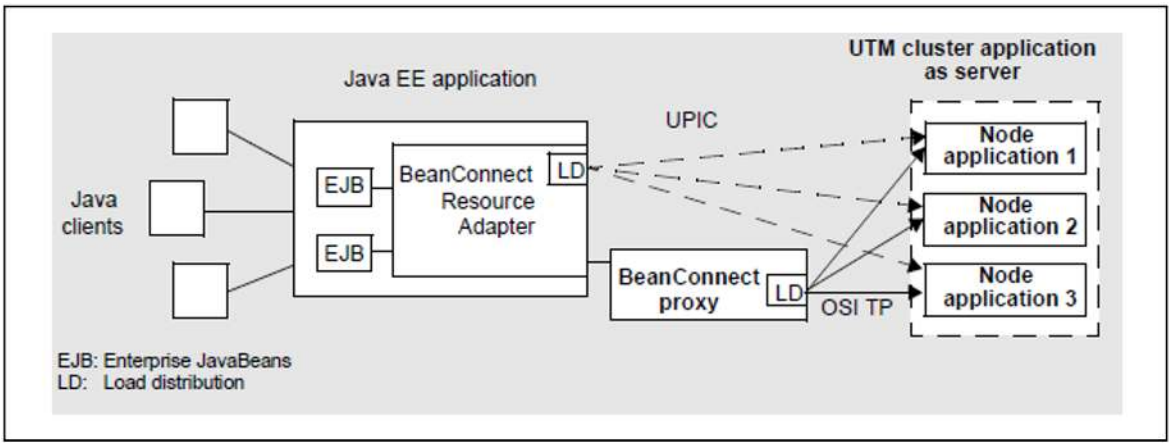


Figure 10: UTM cluster application as server in a Java EE application server

In the case of a connection via OSI TP, load distribution (LD) is performed in the BeanConnect proxy and the round robin method is used.

In the case of UPIC links, the load distributor in the BeanConnect Resource Adapter selects the connections on a random basis.

3.3 Accessing UTM Applications from the Internet

A UTM application can act as an HTTP server and communicate with HTTP clients. Communication can be via http or https connections. For openUTM on BS2000 systems, HTTPS connections are routed via a reverse proxy that acts as a TLS termination proxy. In UTM applications on Unix, Linux and Windows systems, HTTP and HTTPS connections are handled directly by separate network processes.

Thus, a UTM application can offer its services to the Internet via a REST API. Any HTTP clients, such as browsers or cURL clients, are able to access these Services.

This offers the possibility to make services of a UTM application available as Web services. In contrast to WS4UTM, a Web service can be called using HTTP protocol alone, i.e. without the usage of the SOAP protocol.

openUTM supports the HTTP methods POST, GET, PUT and DELETE required for a REST API. Such an interface is also called CRUD interface, for Create, Read, Update, Delete. For UTM itself, the semantics of the HTTP methods are transparent. openUTM forwards all HTTP calls to a user program.

openUTM does not evaluate how an application processes a particular HTTP method. It is therefore up to the application to apply an HTTP method according to its intended semantics.

A user can integrate the output of UTM services into e.g. web pages.

Since communication via the HTTP protocol should be stateless, HTTP clients may only call one-step services of a UTM application, that is, an openUTM service can only send one response to an HTTP client and must then terminate with PEND FI. However, such a service may communicate with job-receiving services or may be structured into several subprogram runs or transactions.

To enable users to integrate HTTP clients into their application environment as easily as possible, openUTM can perform a standard handling of HTTP messages, whereby HTTP clients can call simple, existing line mode programs that do not have to be changed.

On the other hand, it is also possible for users to access all parts of an HTTP message in the application programs and to analyze and process the message by themselves.

In addition to a response message, the user program can also optionally set header fields for the HTTP response. openUTM always adds specific standard headers to a response message.

3.4 Addressing openUTM via WS4UTM

The WebServices consulting project package for openUTM (WS4UTM) makes it easy to provide a UTM application as a Web service.

Web services are applications accessed via Web servers regardless of the platform and that have a standardized interface (open). Through the use of Web service technology, a simple, independent, XML-based, standardized interface is made available to the user by openUTM to access other systems. With the aid of WS4UTM it is possible to make single-step dialog programs in a UTM applications available as Web services (via HTTP/SOAP).

The Apache Axis server is used as the Web service server. Tomcat and Axis (Java) must be installed in order to use WS4UTM.

WS4UTM must be installed on the same computer as the Tomcat/Axis instance used for communication by the WS4UTM.

WS4UTM can be run on the conventional Unix platforms and Windows platforms and consists of the WS4UTMDeploy and WS4UTMAxis components:

- **WS4UTMDeploy** is a graphical tool used to configure and deploy Web services.
- **WS4UTMAxis** is a package of classes loaded by Axis that fulfill the function of a provider. WS4UTMAxis uses then openUTM-JConnect product for calling a service of the corresponding UTM application. The communication is carried out via the UPIC protocol.

The following figure shows the architecture of WS4UTM at runtime:

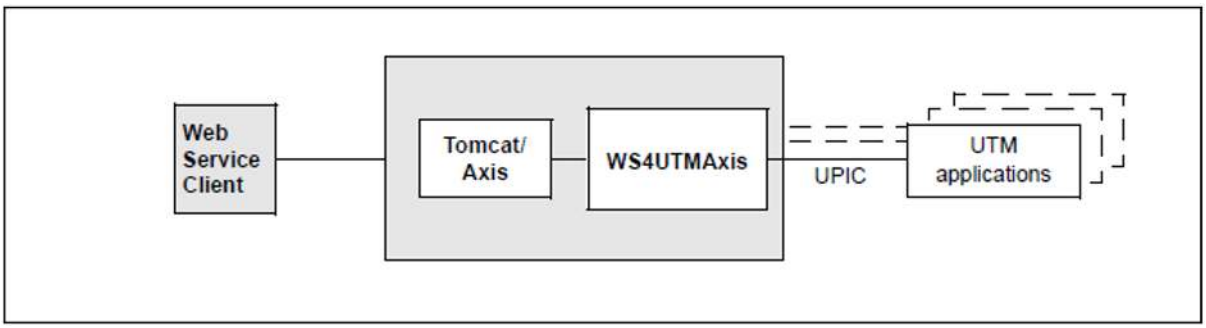


Figure 11: Calling UTM services as Web services with WS4UTM

You will find an overview of WS4UTM in the corresponding Manual.

3.5 Addressing openUTM via WebTransactions

You can connect existing openUTM applications to the World Wide Web unchanged by using the product WebTransactions. The emphasis here is on the term “unchanged”. While the entire server application remains as it is, the presentation on the Web can be designed to suit requirements. Web hosting can take place on BS2000 systems itself or on an independent Web server.

These solutions allow Internet/intranet users to access to the services of the openUTM application server.

WebTransactions for openUTM implements the Web connection of openUTM applications that were developed for FHS or that function via the UPIC client/server interface. WebTransactions for openUTM is available on the BS2000 platforms, Unix platforms and Windows platforms.

WebTransactions runs with any Web server on Unix platforms and Windows platforms. On BS2000 systems, WebTransactions requires the Web server of Apache.

The UTM host application can also run on each of these platforms. The host adapter for communication with the host application is based on UTM-Client (UPIC); in other words, WebTransactions and the host application can run on different platforms.

WebTransactions for openUTM is shipped with the BS2000 program `IFG2FLD`. This program converts the descriptions of the host formats from the IFG library into format description sources. The WebTransactions development environment **WebLab** can then be used to automatically generate templates from the sources. The generated templates form the basis for custom format design.

In addition, WebTransactions offers other opportunities for integration:

- Appearance of the interface (GUIfication)
- Design of the dialog sequences (interface reengineering)
- Application integration (business process reengineering)

In parallel to access via WebTransactions, it is still also possible to access host applications or dynamic Web content via „ordinary“ terminals or clients. This allows you to connection a host application to the Web step by step and to take into account the wishes and needs of different user groups.

You will find an overview of WebTransactions in the manual “WebTransactions - Concepts and Functions”. Detailed information on connecting the different host applications is provided in separate WebTransactions manuals. All WebTransactions manuals are available online in PDF format.

4 Communication with openUTM

This chapter looks at the different forms of communication with openUTM. It deals with client/server communication, server-to-server communication (with and without transaction management) and communication with transport system applications as well as communication via the HTTP protocol.

4.1 Client/server architecture variants

All client/server architectures can be divided into individual software components, known as “layers” or “tiers”. We often speak of 1-tier, 2-tier, 3-tier, and multi-tier models. It is often unclear whether these tiers exist on a physical or logical level:

- Logical software tiers exist when the software is divided into modular components with clear interfaces - irrespective of how these components are distributed in the network.
- Physical software tiers exist when the (logical) software components are distributed across various systems in the network.

In figure 12 on the next page, you will see the five basic client/server modules, each of which has two physical software tiers.

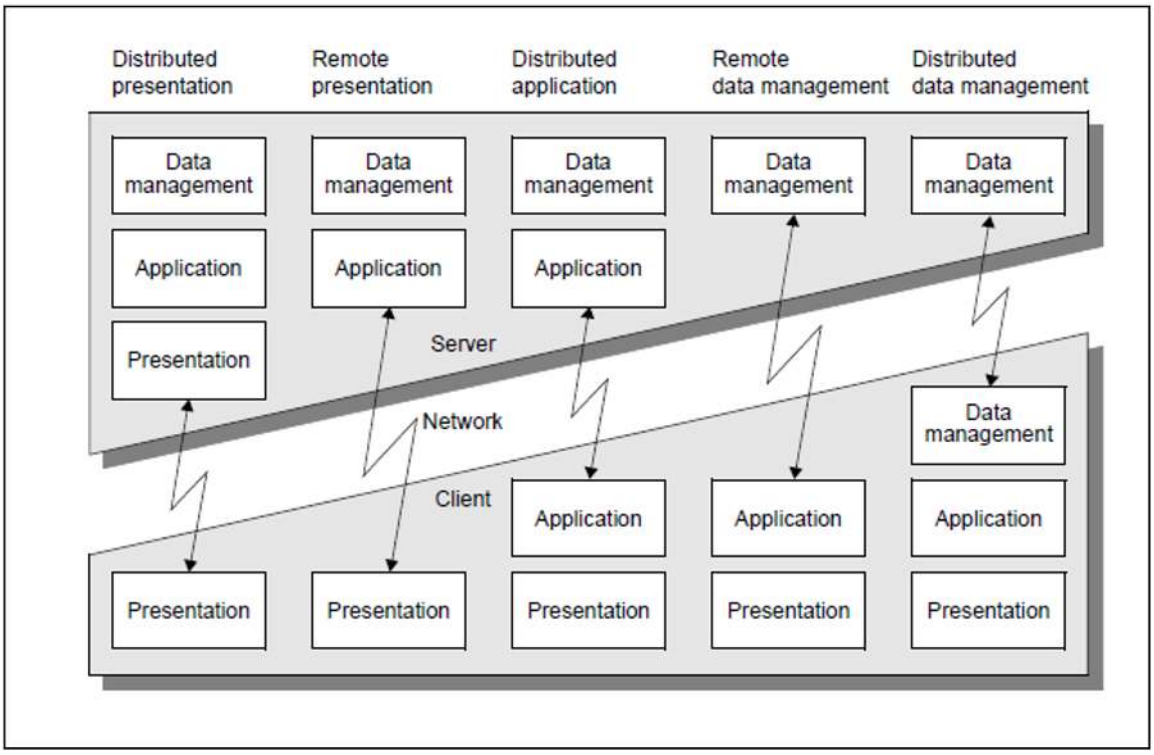


Figure 12: Basic model for client/server architectures

openUTM not only supports the basic model shown in figure 12, but also numerous other adaptations and combinations. It allows for complex multi-tier scenarios on the level of both physical and logical tiers.

As shown in figure 13 (on the next page), openUTM permits the distribution of data `or` application logic, as well as any combinations thereof. Data can thus be stored at the locations at which it is processed. Instead of transmitting large volumes of data via the network, only jobs and results are transferred. This minimizes the network load. openUTM also guarantees global data consistency at all times.

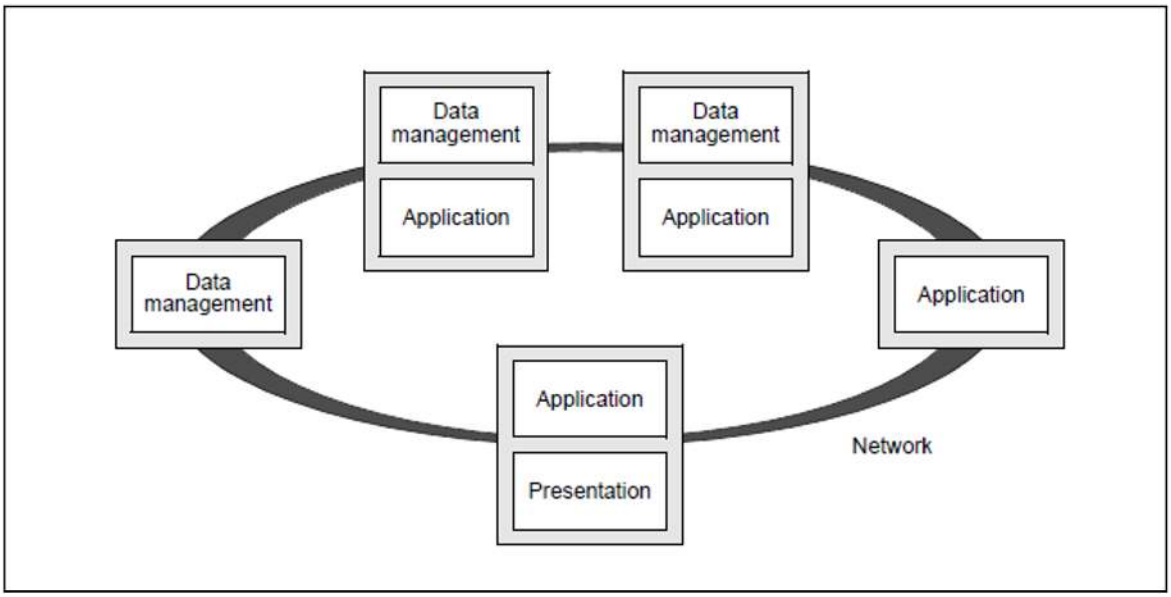


Figure 13: Multi-tier architecture

Disadvantages of conventional 2-tier architectures / advantages of multi-tier distribution

Many conventional client/server applications are based on a classic 2-tier architecture: the server is responsible for data management, while the client performs all other tasks if possible. However, this model has a number of disadvantages:

- Since clients and servers usually communicate on an SQL level, unnecessarily large volumes of data frequently make round trips across the network. In configurations containing many users and when communicating via slow WAN links, the response times are simply unacceptable.
- In this type of architecture, system maintenance is both time-consuming and expensive, since the application logic is implemented on the client systems. This inevitably results in redundancies and maintenance problems, particularly with large applications in heterogeneous environments. This is because each time the application logic is changed, you must perform follow-up maintenance on possibly several source program bases. Since the data model is at least partially visible on the clients, modifications to data management will invariably result in the need for adaptation of the client software.
- Extensions to the application logic will increase the clients' code and resource requirements. This can result in the **fat client syndrome**: the clients must be upgraded, and you may even have to switch over to a more powerful platform.
- To avoid the problems outlined above, many database systems offer the option of relocating at least parts of the application logic to the server by means of stored procedures or trigger functions. This frequently involves the use of proprietary script languages which must be interpreted at runtime, resulting in performance bottlenecks. It is also possible to coordinate several database systems within a single transaction.

Multi-tier architectures based on openUTM allow you to overcome these restrictions and bottlenecks. The logical tiers "Presentation", "Application" and "Data management" have clear physical boundaries and can be distributed arbitrarily. The network load is minimized, the system resources are relieved, and heterogeneous database systems and platforms can be coordinated. The system maintenance costs remain low. When extending the application, the resource requirement increases at only one central location or at a small number of central locations.

4.2 What is meant by the terms “client” and “server”?

Although the terms “client” and “server” are among the most frequently used in the IT world, they are often applied in very different contexts:

In general, they refer to the role assumed by two partners during communication: the **client** requests a service, while the **server** provides the service.

The clients of a UTM application can be:

- terminals or terminal emulations
- UPIC clients (see "Clients with the UPIC carrier system")
- OpenCPIC clients (see "Clients with the OpenCPIC carrier system")
- transport system applications (see "Communicating with transport system applications")
- HTTP-Clients (siehe "Communicating with HTTP-Clients")

UPIC clients and OpenCPIC clients are also referred to as openUTM clients. But the terms “client” and “server” can also refer to entire applications:

UTM applications are known as **server applications**, since they normally act as servers during communication, i.e. they provide **services**. However, when fulfilling certain service requests, they in turn request other services, i.e. they act as clients. Unlike server applications which can assume either role, **client applications** can only function as clients during communication. They are generally responsible for presentation tasks and form the front-end to the users.

Communication between two server applications is known as **server-to-server communication** or **peer-to-peer communication**. This conveys that fact that the communication process involves two partners of equal ranking, although this form of communication can of course distinguish between the client and server roles. This type of interaction is also known as **distributed processing.**

The terms “client” and “server” frequently refer to hardware. Client PCs or client workstations are systems on which client software has been installed. Powerful systems that are particularly suitable for server applications are often called “servers”.

4.2.1 Communication with openUTM-Client applications

Client applications are typically used for presentation purposes, and allow for user-friendly graphical user interfaces. openUTM-Client is a special product provided for developing client applications that use the services of UTM applications. Within these client programs, you can use the X/Open program interfaces CPI-C or XATMI.

For instance, these program interfaces enable you to connect a Visual C++ presentation program on a PC to a UTM application. With this client program, you can input commands for administering a UTM application via a graphical interface on a PC. You can use PC tools to perform statistical analyses by transferring the statistics supplied by openUTM to other PC programs, formatting the statistics graphically, and thus integrating them fully into your Office environment.

The security functions and the easy-to-use restart function in openUTM are also available when you connect client applications.

The openUTM-Client builds on the carrier system. The carrier system’s task is to establish the connection to other components such as the transport access system.

UTM-Clients can be used as a platform for the UPIC carrier system or for the OpenCPIC carrier system. The carrier system you select depends on the type of application. The most important properties of the two carrier systems are described in the following.

4.2.1.1 Clients with the UPIC carrier system

UPIC is a lean, highly efficient and easy-to-use carrier system that is customized for use with openUTM as the server. In UPIC, the client program always initiates communication. The UPIC protocol is used for communication.

A UPIC client can optimally use the UTM functions because

- UPIC supports the transfer of format names and function keys, for example.
- UPIC supports the encryption of system access and user data.
- UPIC can use the SIGNON event service (see "System access control (identification and authentication)" for more information).
- A UPIC client can request the status of the last transaction after a malfunction and is therefore included in the openUTM restart concept.
- A UPIC client can maintain several conversations simultaneously during a program run (“multi-conversations”) as long as the corresponding system supports multi-threading.
- UPIC clients can use the “multi-signon” function. In other words, a number of UPIC clients can sign on at the same time using the same UTM user ID if service restarting is dispensed with for this user ID.
- A UPIC client can establish several parallel transport connections to a UTM application under the same application name (multi-connect”).
- UPIC provides simple load distribution functionality for communication with UTM cluster applications. The UPIC client communicates with one of the associated node applications. This application is selected at random. For more details, see "High availability and load distribution with UTM cluster applications".
- Workload Capture & Replay can be used to simulate load situations for UPIC clients, see section “Workload Capture & Replay”.

UPIC is available for all major Unix, Linux and Windows platforms and for BS2000 systems.

4.2.1.2 Clients with the OpenCPIC carrier system

The OpenCPIC carrier system is more powerful and also more complex than UPIC. OpenCPIC also allows the server program to initiate communication. The OSI TP protocol is used for communication.

An OpenCPIC client can also communicate with OpenCPIC applications, CPI-C applications and other non-UTM applications as long as these applications also use OSI TP. An OpenCPIC client can also work together with a resource manager via the XA interface. openUTM handles communication with openCPIC clients like server-to-server communication.

In communication with OpenCPIC clients, you can select whether to work with or without global transactions:

- In the case of global transaction management, the OpenCPIC client can decide when a global transaction is to begin and end and is therefore included in the global transaction boundary. Transaction control is handled via the X/Open interface TX, while communication is handled via the CPI-C interface.
- When you work without global transactions, the OpenCPIC client is included in the openUTM restart concept.

OpenCPIC clients can use the “multi-signon” function. In other words, a number of OpenCPIC clients can sign on under the same UTM user ID if service restarting is dispensed with for this user ID or if you work with global transaction management.

There are clients that use the OpenCPIC carrier system on Unix platforms and Windows platforms.

Further information on how to use the openUTM security functions when connecting client applications can be found in chapter “Security functions”. You will also find more information on the Restart functionality in chapter “Fault tolerance and the restart function”. The openUTM-Client products UPIC and OpenCPIC carrier systems each have their own manual. There you will find all the details required regarding programming and connecting client applications.

4.2.2 Java clients

The openUTM-JConnect product included in the BeanConnect product provides Java classes that you can use to create clients written in Java. You can connect these clients to UTM applications as UPIC clients, see also section “Integrating openUTM in the Java Enterprise environment”.

There are three ways to do this:

- by means of servlets running on the Web server
- in the browser environment by means of applets
- by means of a direct connection

The following diagrams illustrate these three methods:

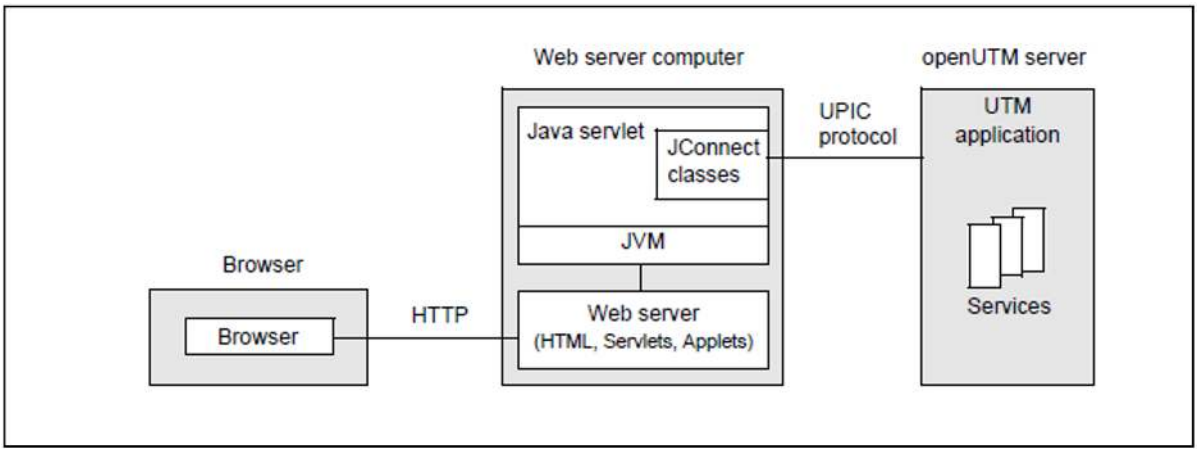


Figure 14: Connecting via servlets running on the Web server

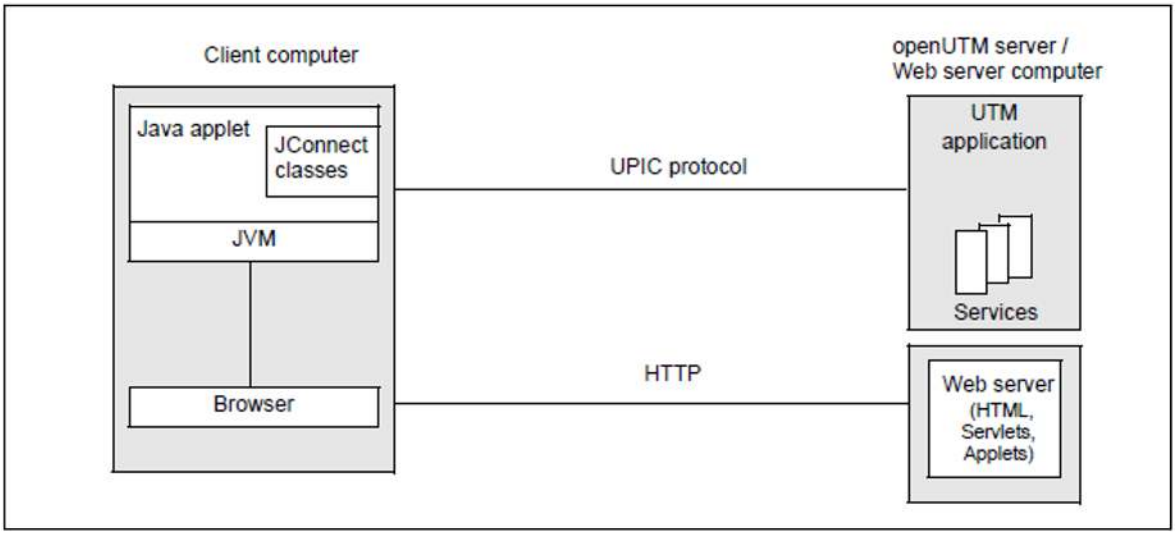


Figure 15: Connecting in a browser environment via applets

An applet is loaded from the Web server via the browser and started on the client computer. This applet then communicates directly with the UTM application via the UPIC protocol. For security reasons, the Web server in this case (figure 15) runs on the same computer as the UTM application, i.e. applets may only communicate with the computer from which they were downloaded.

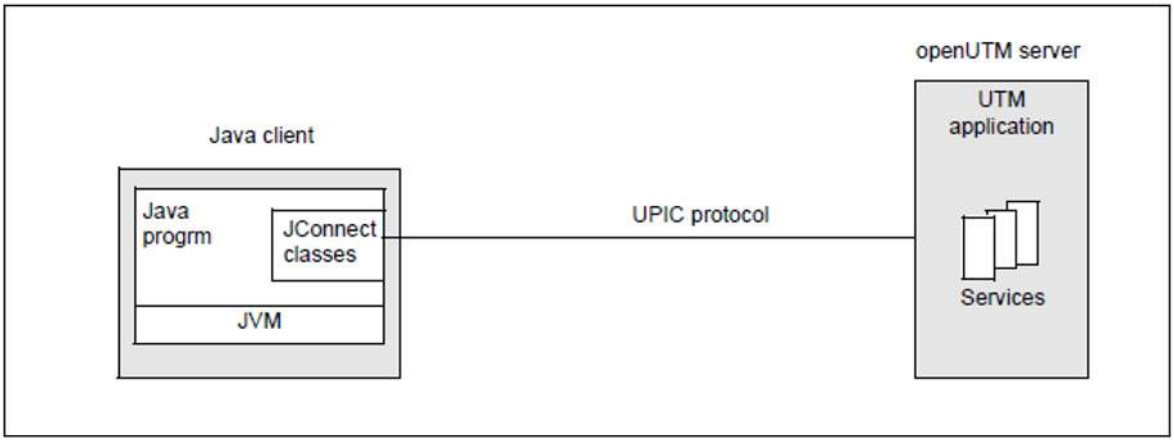


Figure 16: Direct connection of Java clients

The openUTM-JConnect product offers the proven functionality of the product openUTM-Client for the UPIC carrier system in a Java environment, such as:

- conversation with UTM applications
- multi-threading (i.e. a number of parallel conversations)
- support of restart functions
- support of UTM system and data access control
- passwords can be changed by users
- adaptation of the ASCII-EBCDIC conversion

Java programmers are thus offered powerful access to UTM applications with a high level of security.

The JConnect classes are described in the Java docs of the openUTM-JConnect product.

4.3 Server-to-server communication

To fulfill a service request, a server application can in turn request the services of other applications, i.e. it can send its own jobs to other applications. This is also known as distributed processing. With server-to-server communication, therefore, services in two or more applications work together to process a job issued by a client. A service that requests a service from another application is known as a **job-submitting service**, and the service that provides the service is called a **job-receiving service**.

Server-to-server communication also allows you to set up several parallel connections between two applications and process several jobs simultaneously.

An application can not only engage in dialog with other applications, but can also use the openUTM message queuing functionality during server-to-server communication (see section “Sending background jobs to remote services (remote queuing)”).

Since server-to-server communication involves much more than the mere exchange of messages, special high-level communication protocols are required. openUTM supports the LU6.1 protocol and the internationally standardized OSI TP protocol. These protocols, which are widely used worldwide, offer the advantage of enabling a UTM application to interact not only with other UTM applications, but also with applications from third party manufacturers, e.g. CICS, IMS or Tuxedo applications. This applies even if these applications are running on other platforms.

The product openUTM-LU62 permits a dialog with applications that use the LU6.2 protocol. openUTM-LU62 converts the OSI TP protocol to the LU6.2 protocol because openUTM communicates in this case by means of OSI TP (see also section “Communication with CICS, IMS and TXSeries applications”).

Server-to-server communication is fully compatible with cluster operation, i.e. one or both server applications can be implemented as a UTM cluster application, see also section “Load distribution for distributed processing”.

4.3.1 Global dialogs

Within a global dialog, the job-submitting service and the job-receiving service run synchronously, and not separately as with message queuing (see "Message queuing"). This applies irrespective of whether the job-submitting service was itself started within a dialog or via the message queuing system.

Global dialogs allow for complex structures:

- A job-submitting service can communicate with several job-receiving services within a transaction.
- A job-receiving service that processes part of a job for another job-submitting service within a dialog can in turn request a third dialog service in another application.

Such parallel, nested structures result in multi-layered hierarchical relationships which can be represented in tree diagrams. This is known as a **service hierarchy**.

The diagram shows how a service can function simultaneously as a job-receiving service and a job-submitting service. From the point of view of service A, services B, C and D are job-receiving services. From the point of view of services E, F and G on the lowest level, services B and C are job-submitting services.

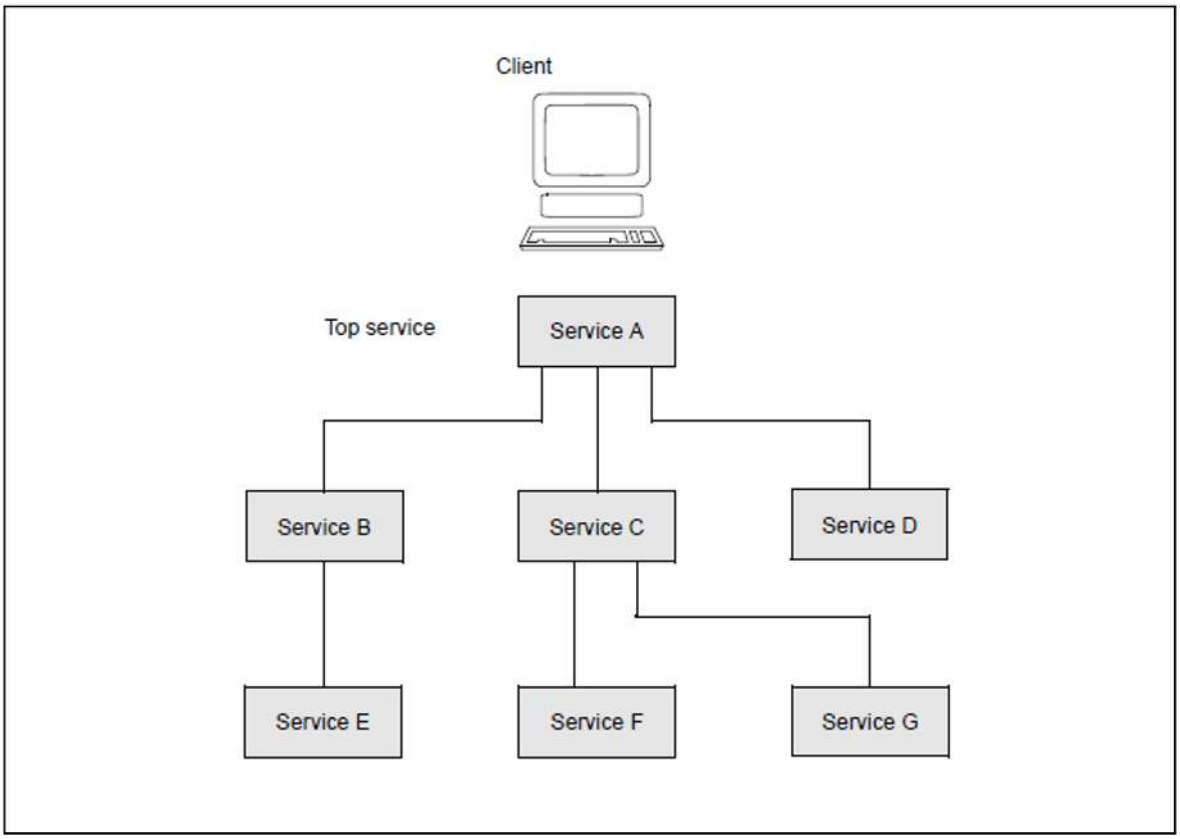


Figure 17: Service hierarchy in global dialogs

Each service hierarchy has a top service. The hierarchy exists only as long as the top service is active. By addressing new job-receiving services and terminating existing job-receiving services, a service hierarchy changes as the dialog job is processed.

Programming global dialogs

In global dialogs, a task is performed by several program units in various applications. A program unit can be addressed from a terminal, a client program, a program from the same application, or a remote application. Depending on the partner, it must therefore decide on the task it performs, where to send messages, and whether or not it is to terminate a global transaction.

openUTM provides a simple, reliable means of creating such complex forms of program-to-program communication through its user-friendly options for controlling global dialogs.

Detailed information on programming global dialogs and on the control options available see openUTM programming manuals “Programming Applications with KDCS for COBOL, C and C++” and “Creating Applications with X/Open Interfaces”.

4.3.2 Transaction management in server-to-server communication

UTM services process jobs in transaction-oriented mode, i.e. an openUTM service consists of one or more transactions. If an openUTM service communicates with another application via OSI TP, it is possible to define whether processing in the remote application is to be included in the transaction or takes place independently of the transaction. The former case involves a distributed transaction, while the latter case involves independent transactions. Distributed transactions are always used when working with the LU6.1 protocol.

Distributed transactions

During server-to-server communication, several local transactions in various applications are involved in processing a job. In the case of global (= cross-application) transaction management, openUTM guarantees global data consistency at all times. For this purpose, openUTM synchronizes the end of the transaction: if the transaction is completed successfully, the synchronization points are set simultaneously in all applications involved. In the event of an error, openUTM ensures that all transactions affected are rolled back. The synchronized transactions thus form a unit, known as a “distributed transaction”. For synchronization purposes, openUTM uses the two-phase commit protocol: as soon as processing of a local transaction is complete, the transaction is initially switched to the “Prepare-to-Commit” state. The global end of transaction (commit) is not set until all the transactions involved have reached the “Prepare-to-Commit” state. An example of this can be found in section “Example: Global dialog with a distributed transaction”.

Asynchronous jobs (MQ jobs) are transferred only once during server-to-server communication with global transaction management. This means that asynchronous jobs are neither lost, nor duplicated even after a power failure or an application crash.

Global transaction management is required in cases whether data consistency and data security requirements are high.

Independent transactions

Unlike distributed transactions, each application is responsible for comitting and rolling back their own independent transactions. In the event of communication errors, this may result in data inconsistencies in the various applications. With this form of communication, there is no guarantee that asynchronous jobs will not be duplicated.

This type of interaction between applications is useful for cases where the data of only **one** application is modified during processing. For example, this applies if one of the applications is a pure retrieval application. Communication is thus more efficient, since the transactions need not be synchronized in the applications involved.

4.3.3 Example: Global dialog with a distributed transaction

A customer wishes to withdraw a cash amount from a particular branch of his bank. However, his account is held at another branch. Each branch uses their own UTM application for data management.

When the customer makes his withdrawal, data in both UTM applications must be updated, i.e. the cashier's balance in one application and the customer account balance in another. This is achieved by means of a single dialog transaction distributed across both applications (distributed transaction). The individual cash transactions (cashier and customer account) are posted in local dialog services. The diagram shows how these local dialog services communicate with each other.

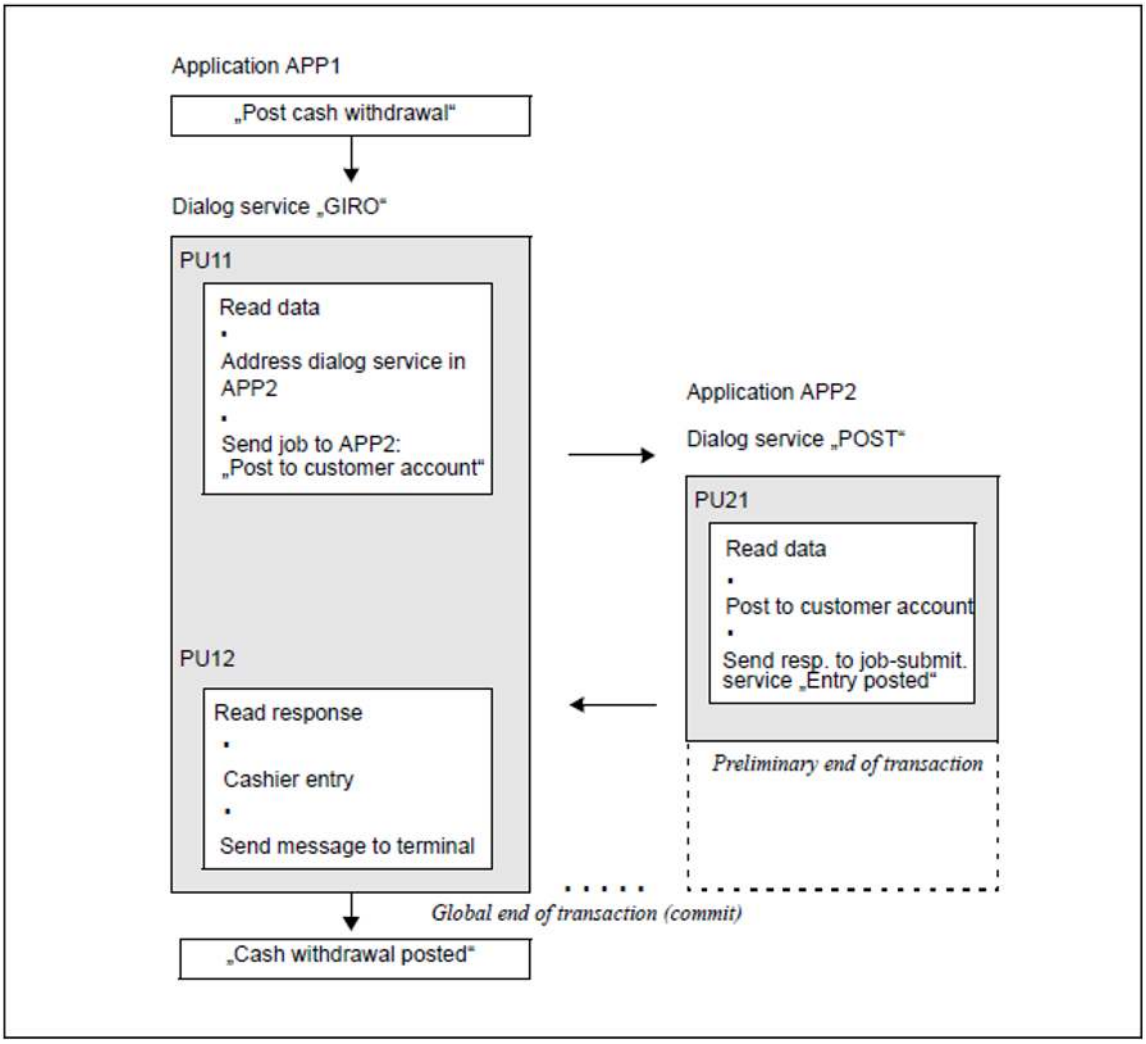


Figure 18: Dialog job with global transaction management

The "GIRO" service in the UTM application APP1 functions as the **job-submitting service**. This service addresses the **job-receiving service**, i.e. the "POST" service in the UTM application APP2, in the dialog program unit PU11.

The program unit PU11 completes its processing step but leaves its transaction open. The program unit PU21 in the "POST" service receives data by means of a dialog message. PU21 returns a dialog response to the job-submitting service and then requests an end of transaction from openUTM.

This dialog response triggers the follow-up program unit PU12 in the “GIRO” service, which terminates the posting job and then requests an end of transaction. openUTM synchronizes the two local synchronization points (= end of transaction) and sets a common synchronization point. The two local transactions (in application APP1 and application APP2) thus form a synchronization unit, in other words a distributed transaction.

If an error occurs before the common synchronization point is set, the entire synchronization unit, i.e. the two local transactions, is rolled back.

4.3.4 Addressing remote services

Before a remote service can be requested, it must be addressed. This is achieved by means of an addressing call in the service routine of the job submitter. In the call, the jobsubmitting service specifies the logical name of the remote service and assigns an identifier to the remote service. This service identifier appears in the job-submitting service for all jobs sent to the remote service, and when reading the results.

The remote service and the remote application are always addressed by means of their logical names. The logical names for remote applications (LPAPs or OSI-LPAPs) and for remote services (LTACs) are defined during generation, and are linked to the actual names in the partner applications. The logical service name is similar in function to the service transaction code. It can be linked to a partner application in one of two ways:

- By generation
  This is known as **single-step addressing**, since the partner application need not be specified in the addressing call.
- In the addressing call in the program
  This is known as **double-step addressing**. This type of addressing is practical in cases where the same service can be started in several applications.

The two procedures for addressing a remote service are illustrated in the following diagrams. The precise format of addressing calls can be found in the openUTM manuals “Programming Applications with KDCS for COBOL, C and C++” and “Creating Applications with X/Open Interfaces”.

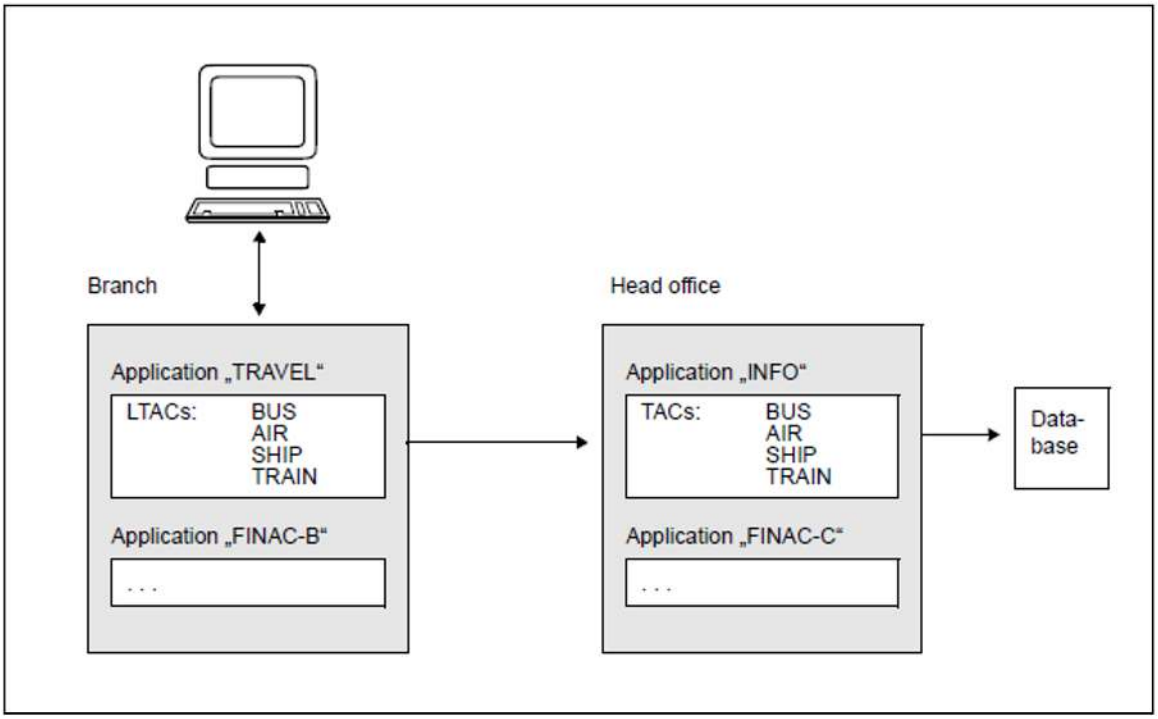


Figure 19: Single-step addressing

The branch offices of a travel agency use the UTM application “TRAVEL”, which retrieves the required information from the UTM application “INFO” in head office. The individual services with the logical names “BUS”, “AIR”, “SHIP” and “TRAIN” were linked to the “INFO” application during generation. It is therefore sufficient to specify the logical service name in the program. The logical name of the partner application need not be specified in the program.

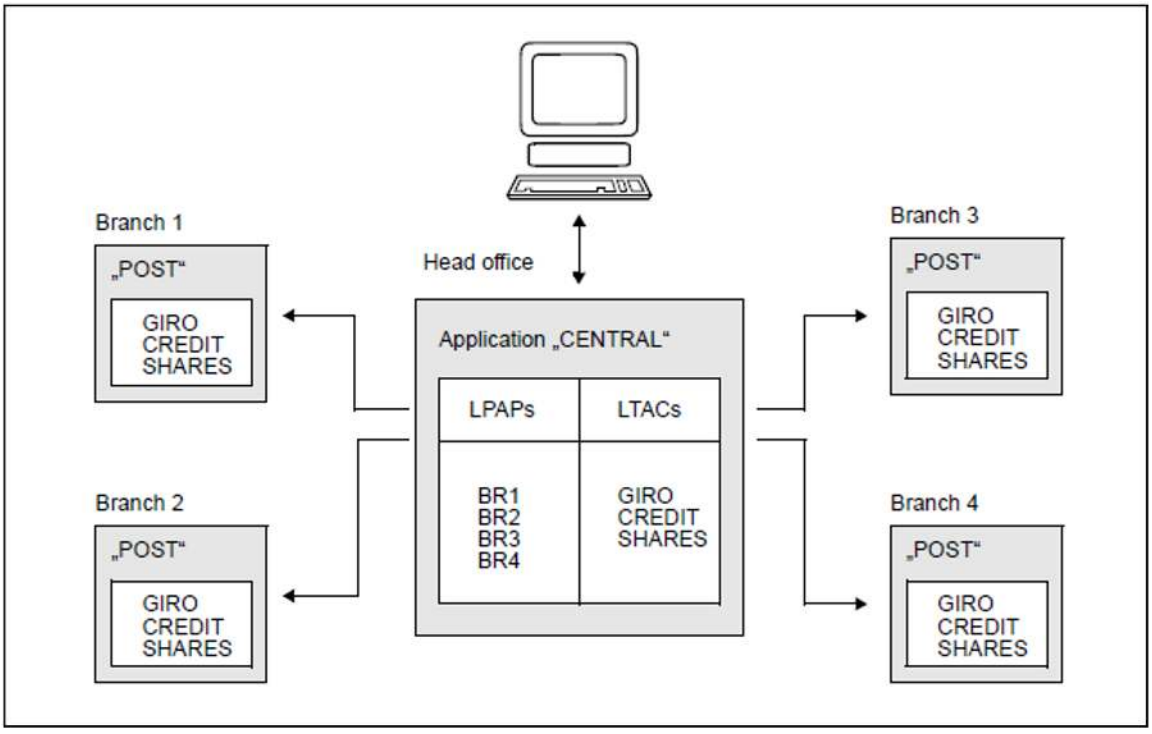


Figure 20: Double-step addressing

Each of the four branches of a bank use the “POST” application. The “CENTRAL” application in head office can access various services, which are identical in all branches. The job-submitting service in head office must therefore select both the service and the partner application in the program.

4.3.5 Communication with CICS, IMS and TXSeries applications

openUTM can communicate with CICS, IMS and TXSeries partner applications via the SNA protocols LU6.1 and LU6.2. The LU6.2 protocol should always be used for new links.

openUTM-LU62 is also required to communicate via LU6.2. openUTM-LU62 is an OSI TP partner from UTM's point of view and an LU6.2 partner from CICS or IMS point of view. With this type of link the partners can work with or without global transaction management.

Refer to the following diagram for the most important types of links.

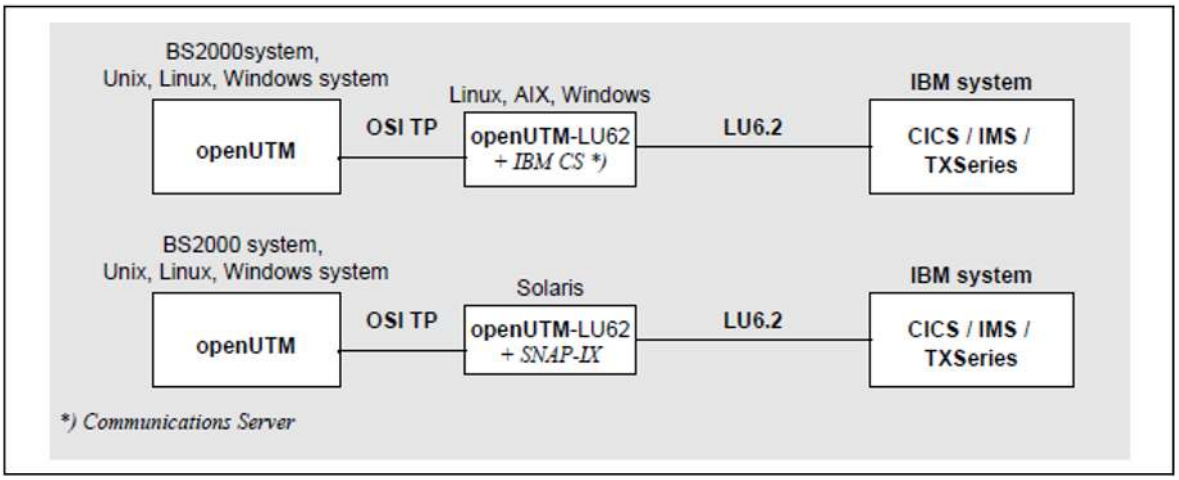


Figure 21: Connecting openUTM to IBM transaction monitors

No additional software is required in the IBM host, the SNA connection is created by the openUTM-LU62 transport system together with the platform-specific components SNAP-IX or IBM Communications Server. If openUTM runs on Unix, Linux a Window system openUTM-LU62 and SNAP-IX or IBM Communications Server can also be installed on this system, and the system can be connected directly to the SNA network.

Job submitter and job receiver

A UTM application can contain job-submitting services as well as job-receiving services when connected to CICS or IMS.

These two roles do not have the same authorizations in openUTM, i.e. certain rules apply to the job-submitting and job-receiving services when a global transaction is terminated. These rules must be observed by the CICS and IMS application programs, even if the programs allow a different procedure to be used when this occurs.

openUTM as the job submitter

A UTM job-submitting service addresses a CICS or IMS job-receiving service exactly like it does a UTM job-receiving service. openUTM can submit dialog jobs as well as asynchronous jobs (message queuing).

CICS as the job submitter

CICS can start dialog services as well as asynchronous services in openUTM.

IMS as the job submitter

The possibilities depend on the type of connection in this case:

- For LU6.2 interconnection, IMS can start dialog services as well as asynchronous services in openUTM.

- For LU6.1 interconnection, on the other hand, IMS can only start asynchronous services in openUTM. In order to allow IMS to open dialogs in spite of this, LU6.1 provides the ability to create dummy dialogs between asynchronous services. A dummy dialog returns additional information to the job submitter that can be used by the job receiver to send the reply back to the correct partner. openUTM contains a special extension of the program interface specifically for this purpose.

  Dummy dialogs are not possible in conjunction with CICS partners.

You will find a detailed description of how to generate and program connections with CICS and IMS in the manual “Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications”.

4.4 Communicating with transport system applications

In addition to transaction-oriented distributed processing, which requires high-level communication protocols, a UTM application can also communicate with applications running directly above the transport system interface without global transaction management. Examples of such applications include the CMX applications on Unix, Linux or Windows systems, DCAM applications on BS2000 systems or any TCP/IP socket applications. On socket connections, UTM supports the HTTP protocol as well as the UTM-specific UTM socket protocol (USP). In the following, socket applications that communicate with the UTM application via HTTP are referred to as HTTP clients (see chapter "Communication with HTTP Clients"), while transport system applications that communicate with the UTM application via USP are referred to as socket USP applications.

Since high-level protocols are not used, support for global transactions cannot be provided when communicating with transport system applications. In this case, openUTM can only provide local transaction processing capability.

In the event of errors in the transport system or an abnormal termination of the UTM application, there is no guarantee that a message sent to another application will be received or processed by the target application. It is thus possible for messages to be lost or duplicated.

The following sections contain some useful information regarding interaction between a UTM application and transport system applications.

Connection establishment

The initiative for establishing a connection may come either from the other application or from the UTM application. When communicating with HTTP clients, the HTTP client always initiates the connection establishment.

Establishing a connection to the UTM application

If the transport system application establishes the connection, then it is signed on to the UTM application using a user ID permanently assigned to the partner, also called the connection ID.

If the UTM application has a user-written sign-on dialog (SIGNON event service, see "System access control (identification and authentication)"), then this dialog is also run when transport system applications (except HTTP-clients) sign on. This sign-on dialog can be used, for example, to assign the transport system application a real user ID for subsequent processing.

Transport system applications can use the “multi-signon” function, i.e. several transport system applications can sign on under the same real UTM user ID at the same time, but only if the service restart functionality is not used for this user ID.

A transport system application can establish several parallel transport connections to a UTM application under the same application name (“multi-connect”).

Processing jobs

A transport system application (not equal to HTTP client) can send both asynchronous and dialog jobs to the UTM application. An HTTP client can only send dialog requests to the UTM application. However, the following conditions must be observed:

- The Jobs of a transport system application (not equal to HTTP client) must be submitted in the format expected by openUTM, i.e. the first eight characters of the message must contain the transaction code under which the service to be started was generated for openUTM. This transaction code enables openUTM to determine whether the service is a dialog or asynchronous service.

- Socket USP applications send a byte stream, while openUTM works on a message-oriented base. To enable the UTM application to recognize message boundaries, a socket USP application must use the UTM socket protocol (USP) from which the length of a message can be recognized. The USP allows messages to be sent in several parts, which means that the entire message can be larger than the size of the input buffer. The USP can also be used for output, as well, if you choose.
- HTTP clients also send a byte stream. HTTP clients also send a byte stream that is described by the HTTP protocol. openUTM analyzes the HTTP protocol and transfers the query and the message body of the HTTP request to the user program as message parts. You can use an HTTP exit to split the message into several message parts for a program unit.
- In the case of dialog jobs, you must adhere strictly to the dialog format required by openUTM, i.e. the partner application must wait to receive a response from the UTM application before sending the next message.
- openUTM does not send any messages of length 0 to transport system applications. Although such messages are not actually sent, they do switch send authorization from one application to the other. openUTM then waits for a message from the transport system application. With multi-step dialogs, it is therefore important to observe the logic of the dialog sequence. In principle it is not possible to send messages with a length of 0 to HTTP clients.
- In communication with HTTP-clients and non-socket partners, message lengths are limited by the size of the input/output buffer.
- When communicating using the USP protocol, the messages can be fragmented (see the note above on the USP). Only the length of fragments is limited; there is no limit on the length of the entire message.
- openUTM does not format messages intended for other applications, i.e. the target application receives the message in the format in which it was supplied to the message area by the program unit.
- Automatic code conversion (ASCII-EBCDIC) can be initiated at generation.
- Restart capability: if necessary, dialog services can be restarted. During a service restart, the last output message is repeated.

While running a UTM application, messages may occur which refer to the communication partner of the UTM application. Only those UTM messages for which the destination PARTNER was specified in the UTM message module are dispatched to the communication partner. By creating a local message module, this destination can be added to or removed from other messages (see the corresponding openUTM manual ”Messages, Debugging and Diagnostics”). These UTM messages can occur within or outside a dialog. The communication partner must be able to respond to the messages accordingly. To HTTP-clients only messages K017 and K034 are sent, regardless of whether the message target PARTNER has been specified.

If UTM user commands are issued by another application, these are not interpreted by openUTM as commands.

Connection shutdown

The initiative for shutting down a connection may come from the transport system application or from the UTM application. If the connection is to be shut down by the UTM application, this can be achieved either by entering an administration command directly or by using an administration call or a KDCS call (SIGN OFF) from a program unit run. For HTTP clients, UTM closes the connection after each service, if the connection header in the HTTP request is not set to "Keep Alive".

You will find more information on interconnection with socket-USP-applications in the openUTM manual “Generating Applications”.

4.5 Communicating with HTTP Clients

In addition to the descriptions in the chapter "Communicating with transport system applications", this chapter contains information on communication with HTTP clients.

- Structure of an HTTP message
- Adressing of a program unit
- Generation
- HTTP Header
- Event Exit HTTP
- Usage scenarios

4.5.1 Structure of an HTTP message

An HTTP message consists of a request or status line, header fields and, optionally, a message body. A payload intended for a user program can be included in the query segment of the request line and in the message body. The header fields contain information about the properties of the message.
An HTTP client addresses a service using a URL (see RFC 3986). A URL is structured as follows:

```
http://example.com:8042/over/there?name=ferret#nose
\_/    \______________/ \________/ \_________/ \__/
 |            |             |          |        |
scheme     authority        path      query    fragment
```

With openUTM, the path of a URL always starts with the first slash ("/") after the authority part.

When calling a service of a UTM application via an HTTP request, a user should not specify a fragment of the URL, since many HTTP clients remove the fragment from the request and do not transmit it to the server. In an HTTP response, fragment information is used by the client to interpret the response, for example, to position it on a specific page of a pdf file or on a specific chapter of an HTML message.

4.5.2 Adressing of a program unit

The path of an HTTP request is used to address a resource. openUTM uses the path information to determine the TAC and therefore the program unit to which an HTTP request is to be sent.
A user has various options for specifying which TAC openUTM is to call for an HTTP request with a particular path.

One way this can be done is using the HTTP-DESCRIPTOR generation statement. This statement allows

- to define the mapping of the path to a TAC and
- to supply additional parameters to influence the handling of such a request by openUTM.

For example, you can specify that openUTM on BS2000 systems should convert the message into the locally used EBCDIC code, or that openUTM should call an additional user program, called an HTTP exit, before calling the TAC. This user program can, e.g. transform the message from the format used by an HTTP client into the format expected by the addressed program unit.

When you specify the path in the HTTP-DESCRIPTOR generation statement, you can also use a wildcard character that allows you to easily assign different path specifications that match at the beginning to the same TAC.

Another way of calling a program unit of a UTM application via an HTTP request is to write the TAC of the program unit directly in the path information.

Example:
`http://<host:port>/KDCINF?STAT`

By this request the TAC KDCINF is called, provided the caller has the authorization to use this TAC (see also the USER-AUTH operand of the BCAMAPPL generation statement). The character string "STAT" is transferred to the program unit as data.

openUTM tries to map the path directly to a TAC whenever the path was not specified in an HTTP-DESCRIPTOR statement.

4.5.3 Generation

The important instructions for communicating with HTTP clients are:

- BCAMAPPL
- TPOOL
- HTTP-DESCRIPTOR
- CHAR-SET (BS2000-Systeme)

BCAMAPPL

You can use a BCAMAPPL statement to generate a transport system end point for HTTP clients in the UTM application.

```
BCAMAPPL <bcamname>

        ,LISTENER-PORT = number
        ,SIGNON-TAC    = *NONE
        ,T-PROT        = (SOCKET, *HTTP | *ANY, [SECURE])
        [,USER-AUTH    = *NONE | *BASIC]
```

In the LISTENER-PORT parameter, you specify the port number that HTTP clients must use to address the UTM application.

A BCAMAPPL statement intended for access by HTTP clients must be generated without a logon procedure, that is, *NONE must explicitly be specified for the SIGNON-TAC parameter.

In the parameter T-PROT you must specifiy *HTTP or *ANY as a variant of the socket protocol. By specifying SECURE, you can define that communication has to take place via SSL connections.

The USER-AUTH parameter can be used to specify that a user must log on to the UTM application with his or her UserID and password.

TPOOL

In order for an HTTP client to be able to connect to the UTM application, a logical connection point called a LTERM partner must be generated. You can use a TPOOL statement to generate a pool of LTERM partners for HTTP clients. The TPOOL must be generated with PTYPE=SOCKET, MAP=USER and USP-HDR=NO.

```
TPOOL BCAMAPPL  = <bcamname>

        ,LTERM   = <prefix>
        ,MAP     = USER
        ,NUMBER  = number
        ,PRONAM  = *ANY | <subnet>
        ,PTYPE   = SOCKET
        ,USP-HDR = NO
        ,ENCRYPTION-LEVEL = <level>
```

If the BCAMAPPL is generated with T-PROT=(Socket,...,SECURE) then the value for ENCRYPTION-LEVEL is implicitly set to TRUSTED, that is, HTTP clients that connect via this TPOOL are considered trustworthy.

HTTP-DESCRIPTOR

The HTTP-DESCRIPTOR statement can be used to map the path specification of an HTTP request to a TAC of the UTM application.

```
HTTP-DESCRIPTOR <http-descriptor-name>
      [,BCAMAPPL     = <bcamappl> | *ALL]
      [,CONVERT-TEXT = *YES | *NO]
      [,HTTP-EXIT    = <program-name> | *NONE | *SYSTEM ]
       ,PATH          = C'<path1..254>'
       ,TAC           = <tac>
      [,USER-AUTH    = *NONE | *BASIC]
```

The parameter BCAMAPPL can be used to determine whether the assignment of path to TAC should only be valid for HTTP requests that arrive via a particular transport system end point, or whether they should be valid for all HTTP connections regardless of the transport system end point.

In BS2000 systems, the parameter CONVERT-TEXT can be used to specify whether or not openUTM is to perform a code conversion of the query and the message body of an HTTP request.

In the HTTP-EXIT parameter, the user can specify a program that formats HTTP messages for an existing program unit. openUTM then calls this program for incoming and outgoing HTTP messages. With *SYSTEM, you can specify that openUTM should convert output messages to HTML format.

In the PATH parameter, you define the path that is to be assigned to the TAC specified in the TAC parameter. The path specification may contain a "*" as the last character, which is then evaluated as a wildcard character. This makes it easy to assign path specifications that are identical in their leading part to the same TAC.

In the USER-AUTH parameter, you can specify whether HTTP requests that are assigned to this HTTP descriptor must contain authentication data. If the value *BASIC is specified here, the client must provide information about the UserId and password in the authorization header of an HTTP request. If this is not the case, then openUTM requests this information by sending of a response with status code "401 Unauthorized".

CHAR-SET (BS2000 systems)

The protocol segments of an HTTP message (incl. query) are always encoded in US ASCII and are converted to EBCDIC by openUTM in BS2000. The user message can be encoded in different ways. This means that, at least in BS2000 systems, segments of an HTTP message must be encoded according to EBCDIC.

In BS2000 systems, the generation statement CHAR-SET can be used to assign Character Set names to the code conversion tables SYS1, ..., SYS4 of openUTM.

```
CHAR-SET { SYS1 | SYS2 | SYS3 | SYS4 }

             ,NAME = (C'<char-set-name1..32>', ...)
```

If CONVERT-TEXT=*YES is generated for an HTTP DESCRIPTOR in BS2000 systems, openUTM analyzes the content-type header for HTTP requests that can be assigned to this HTTP descriptor. If a character set defined in openUTM with a CHAR-SET statement is found, openUTM performs a code conversion for the message body of the HTTP request with the conversion table assigned by the CHAR-SET statement. openUTM then also uses the corresponding conversion table for the the message body of the HTTP response.

4.5.4 HTTP Header

openUTM evaluates the following header fields in an HTTP request, whereby the keywords are recognized in any upper-case notation:

- **Accept**
  Specifies the data type in which the HTTP client wants to receive the HTTP response.

- **Accept-Charset**
  Specifies the character set in which the HTTP client wants to receive the HTTP response.

- **Authorization**
  Specification of the authentication data.

- **Connection**
  Specifies whether the connection is to be held by the HTTP server.

- **Content-Length**
  Specifies the length of the message body.

- **Content-Type**
  Specifies the MIME type and, if necessary, the character set of the message body.

- **Expect**
  Request to the HTTP server to check the request before receiving the message body.

- **Host**
  Specifies the name of the HTTP server.

- **Transfer-Encoding**
  Specifies the data transformation used for the request.

openUTM sets the following header fields in an HTTP response.

- **Cache-Control**
  Specifies whether and for how long the response may be stored by the client.

- **Connection**
  Specifies whether the connection is terminated by the HTTP server.

- **Content-Length**
  Length of the message body.

- **Content-Type**
  Specifies the MIME type and, if necessary, the character set of the message body.

- **Date**
  Specifies the current date.

- **Server**
  Information about the UTM HTTP server

- **X-Content-Type-Options**
  Information on MIME sniffing.

- **X-Frame-Options**
  Protection against clickjacking.

- **X-Xss-Protection**
  Filter for cross-site scripting.

You will find detailed information on the evaluation and the setting of header fields during application running in chapter "Program interface UTM-HTTP" in the openUTM manual „Programming applications with KDCS“.

4.5.5 Event Exit HTTP

If the message body of an HTTP message is a text message, it is normally structured in an HTML, XML or JSON format. Since existing program units are not prepared for such a data format, HTTP messages can optionally be transferred to a user exit that is referred to as an event exit HTTP or HTTP exit. The task of the HTTP exit is to convert the network representation of a message to the format expected by a program unit or vice versa.

The event exit HTTP is defined during generation with a PROGRAM statement and assigned to a TAC using the HTTP-DESCRIPTOR statement.

4.5.6 Usage scenarios

The following are four application scenarios for accessing UTM applications from HTTP clients.

Direct call of existing line-mode program units

Existing line-mode programs can be called by HTTP clients without any changes to the application programs being necessary.
The prerequisite is that the addressed operation is a one-step operation.
The message for the program unit can be passed in the query and/or the message body of the HTTP request.

If the TAC of the program unit is specified directly in the path, openUTM performs a code conversion of the HTTP messages for such a TAC in BS2000 systems. The HTTP response formats openUTM as an HTML message, whereby several message parts passed with MPUT NT are combined and line breaks are received.

Call existing program units via an HTTP Exit

This scenario is useful for programs that expect data in a defined layout, such as programs that were previously called by UPIC clients.

The program unit does not have to be changed. Instead, the user can write an HTTP exit program and assign this exit program to the TAC in an HTTP-DESCRIPTOR statement.

The HTTP exit program is then called by openUTM for each input and output message of this TAC.

In the HTTP exit program, the user can perform the conversion of the network message to the data structure expected by the program unit, as well as the conversion of the response message from the program unit into a network message.

The HTTP exit program can also subdivide an HTTP request into several message parts, each of which can then be read with a separate MGET NT call. In a similar way, the HTTP exit program can assemble an HTTP response from several message parts supplied by several MPUT NT calls.

Calling existing program units from new HTTP clients

If an existing application is not to be modified, then a user can create new client programs, that communicate with the UTM application using the HTTP protocol and which provide data in the format expected by the existing program units of the UTM application and present the response of the UTM application in an appropriate form to the user.

Develop new program units for communication with HTTP Clients

New programs created specifically for communicating with HTTP clients have various ways of accessing protocol information.

On the one hand, information about an HTTP request can be obtained by an option of the INIT PU call.

On the other hand, openUTM provides a number of functions that allow to access segments of an HTTP message directly.

Reading an HTTP message

A payload message intended for a program unit can be specified in the query and/or in the message body for an HTTP request.

Reading an HTTP message without HTTP exit

A program unit can read the payload message of an HTTP request with MGET. If the HTTP request contains a query, openUTM returns the query during the first MGET of a program unit run. The message body can be read with subsequent MGET calls.

If no query is contained in an HTTP request, on the first MGET openUTM returns the (first part) of the message body. Subsequent MGET calls can be used to read further parts of the message body.

Reading an HTTP message with HTTP exit

In this case, the HTTP exit determines which data the program unit receives from the MGET and how this data is structured in message segments.

Structure of an HTTP-Response

Response messages to an HTTP client are passed to openUTM with MPUT calls. A user program can also use special functions to set the status code and header fields of an HTTP response.

Functions for accessing parts of an HTTP message

Not only the payload message of an HTTP message may be relevant for user programs. The URL and the HTTP headers may also contain information that an application program needs to process a request.

For this purpose openUTM provides a number of functions that can be used to read or write parts of an HTTP message from an HTTP exit or a program unit. These functions enable a user to control the program flow and supply parts of the response message.

An overview on these functions can be found in the subchapter „Program interface UTM-HTTP" in the openUTM manual „Programming Applications with KDCS".

4.6 Overview: partners, protocols, transaction management

The table below shows the partners with which a UTM application can interact. These are just examples, and this table should not be taken as a complete listing.

Partner	Protocol	Without global transactions	With global transactions
Other UTM applications	LU6.1	-	x
	OSI TP	x	x
	Transport system	x	-
openUTM-Client applications with the UPIC carrier system	UPIC	x	-
JConnect-Client with JConnect classes	UPIC	x	-
openUTM-Client applications with the OpenCPIC carrier system	OSI TP	x	x
Java EE applications	UPIC Transport system	x	-
	OSI TP	x	x
CICS/IMS/TXSeries applications on an IBM mainframe	LU6.1	-	x
	openUTM: OSI TP IBM: LU6.2	x	x
CICS/IMS applications on other IBM systems, e.g. AIX system	openUTM: OSI TP IBM: LU6.2	x	-
Tuxedo applications, applications in UNISYS environments, and all other applications that support the OSI TP protocol	OSI TP (LU6.2 is also possible for Tuxedo)	x	x
DCAM applications	Transport System	x	-
PCMX applications in a remote Unix, Linux or Windows system	Transport system	x	-
Other transport system applications based on OSI transport layers or on RFC1006 via TCP/IP	Transport system	x	-

Partner	Protocol	Without global transactions	With global transactions
Socket-USP-Anwendungen	Transportsystem	x	-
HTTP-Clients	HTTP	x	-

5 Message queuing

- The merits and options of the message queuing functionality integrated in openUTM have already been mentioned briefly in section “Message queuing”.

  For convenience, the main advantages are again listed briefly below:

  - temporal and geographical independence from the communicating components
  - absolutely reliable message transmission guaranteed through the transaction-oriented deferred delivery mechanism
  - independence of the current connection

  Message queuing (MQ) is a form of communication in which messages are not exchanged immediately, rather are buffered in intermediate queues before being dispatched (store and forward). These messages are known as **asynchronous messages**. The communication is processed in the form of **asynchronous jobs**. An asynchronous job consists of the asynchronous message, the recipient of the message and possibly also the desired time of execution.

  A distinction is drawn between UTM-controlled queues and service-controlled queues, depending on who is responsible for processing the messages sent to the queue.

  - In the case of **UTM-controlled queues,** the queuing mechanism is made available in its entirety by openUTM. In other words, openUTM also offers triggering functions in addition to pure queuing functionality.
  - In the case of **service-controlled queues,** a service is responsible for the further processing of the message. In other words, openUTM executes pure queuing functionality.

  The control options described on "Control options for message queues" are available for both output and background jobs:

  - time control
  - messages returned via confirmation jobs
  - queue administration

  MQ calls of the KDCS interface are available for message queuing. These are introduced in section “Message queue calls of the KDCS interface”.

5.1 UTM-controlled queues

In the case of UTM-controlled queues, openUTM carries out all the subsequent processing once the message is placed in the queue. A distinction is drawn between **output jobs** (output queuing) and **background jobs**, depending on the recipient.

5.1.1 Output jobs (output queuing)

Output jobs are asynchronous jobs that output a message (e.g. a document) on a printer or terminal. Messages can also be output to another application connected via the transport system interface (see section “Communicating with transport system applications”).

Output jobs contain information on the output destination and the message to be output asynchronously. They are automatically processed by the UTM system functions without the intervention of the application program.

Output jobs can be triggered by MQ calls from a dialog service or asynchronous service of the UTM application.

5.1.2 Background jobs

Background jobs are asynchronous jobs sent to an asynchronous service of the local application or of a remote application. We therefore also refer to local queuing or remote queuing.They are particularly suitable for long-running or non-time-critical processes, the result of which has no direct influence on the current dialog.

They consist of the transaction code (TAC) of the program unit with which the background job begins, and possibly an asynchronous message. The transaction code determines whether the job is processed asynchronously or as a dialog job.

Background jobs can be initiated by means of:

- input at a terminal
- a call from an UTM-Client program with the OpenCPIC carrier system
- a message from another application that communicates with the UTM application by means of the LU6.1 or OSI TP protocol
- input from another application connected via the transport system interface
- an MQ call from a service of the local application or a remote UTM application
- UTM messages (i.e. event-driven)

Background jobs are redelivery-capable, i.e. they can be restarted after abnormal termination of an asynchronous service and the asynchronous messages can then be redelivered.

Alternatively, or after the last redelivery attempt, openUTM can put the incorrectly processed message of an asynchronous service in a separate queue, the dead letter queue. Messages to an asynchronous service of a remote application via the LU6.1 or OSI-TP protocol can also be put in the dead letter queue if these messages cannot be delivered due to a permanent error in the remote application.

5.1.2.1 Processing background jobs

To process a background job, the UTM application starts the **asynchronous service** at an appropriate time. This service performs all the steps required to complete the job.

An asynchronous service can be divided into several processing steps and transactions. When using the KDCS interface, it can also comprise several program units.

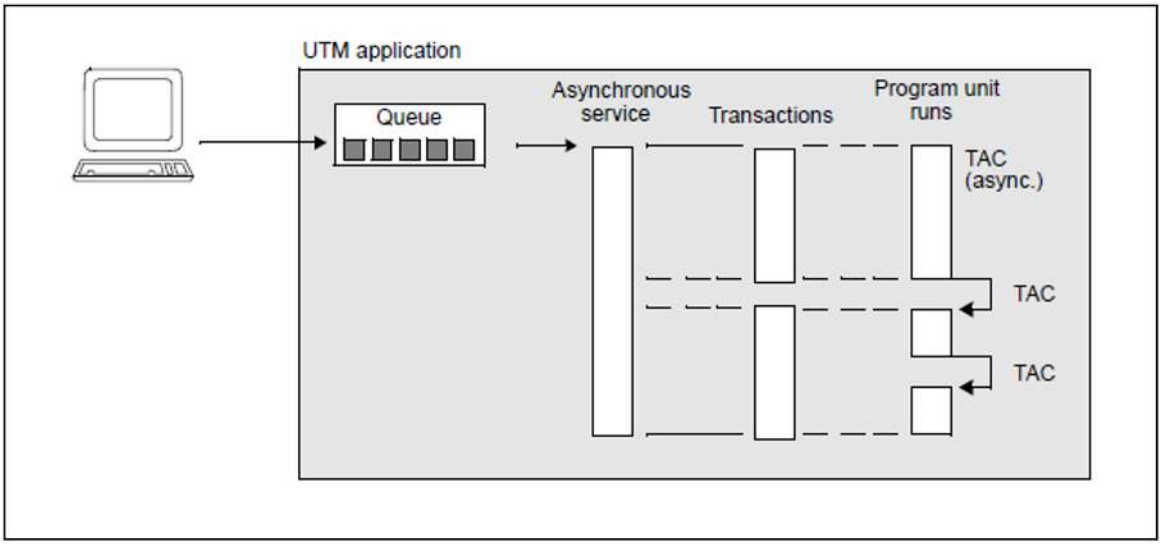


Figure 22: Structure of an asynchronous service initiated from a terminal

In the example in figure 22, a background job is sent from a terminal to an asynchronous service of the local application. The transaction code of the service and possibly a message are input at the terminal. openUTM automatically places the job in the corresponding queue, and starts the asynchronous service independently of the job submitter as soon as the required resources are available.

An asynchronous service can initiate its own asynchronous jobs. These can be output jobs or further background jobs. In the case of distributed processing, dialog jobs can also be sent to partner applications from an asynchronous service, i.e. the asynchronous service can start its own remote dialog services.

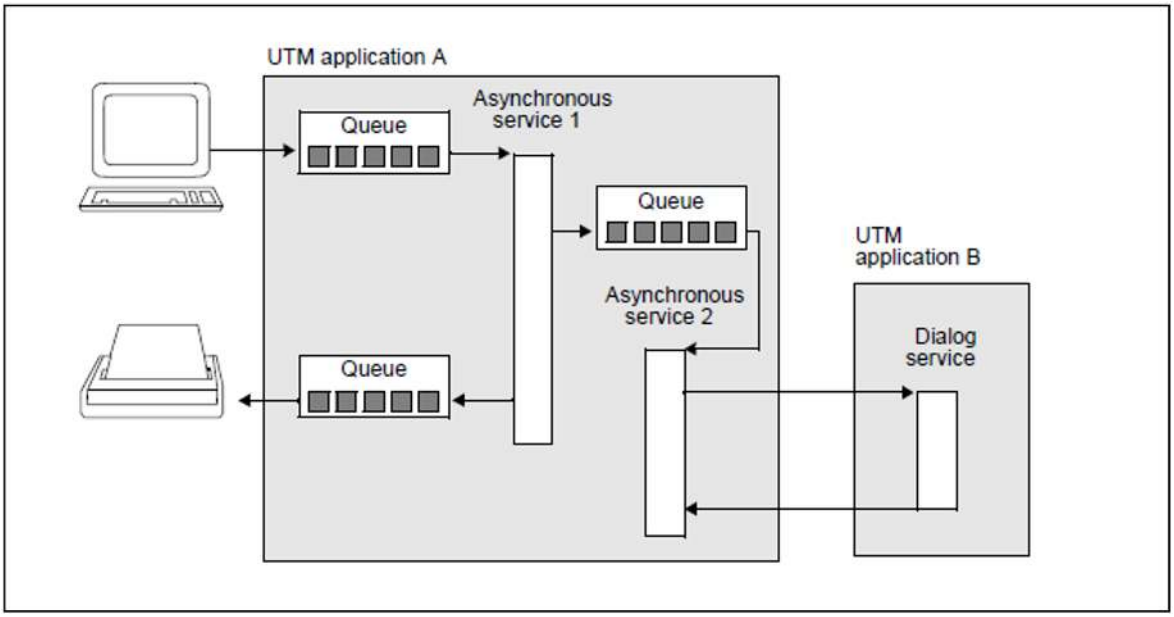


Figure 23: Asynchronous services that initiate their own jobs

In figure 23 - as in figure 22 - a background job is sent from the terminal to a local asynchronous service. This service in turn initiates a further background job, and sends an output job to a printer. Asynchronous service 2 starts its own remote dialog service in UTM application B. This application can be located on the same system as UTM application A, or on a remote system. Even in the case of more complex structures with numerous queues, you need not concern yourself with the queuing mechanism, as this is provided automatically by openUTM.

5.1.2.2 Sending background jobs to remote services (remote queuing)

It is possible to send background jobs not only to asynchronous services of the local application, but also to asynchronous services of remote applications.

This is where one of the main strengths of the openUTM MQ functionality comes to the fore. In the case of background jobs sent to remote applications, openUTM works with two local queues: one in the sender application and the other in the receiver application. Thanks to this deferred delivery principle, distributed message processing with openUTM will continue to operate regardless of whether or not a connection is currently possible. If a connection cannot be established, the job remains in the local sender queue until the connection has been set up.

With remote queuing, you need not concern yourself with the queuing mechanism. You simply specify the asynchronous service for which the MQ message is intended. Remote asynchronous services are addressed in the same way as remote dialog services. This simplifies the design of distributed applications.

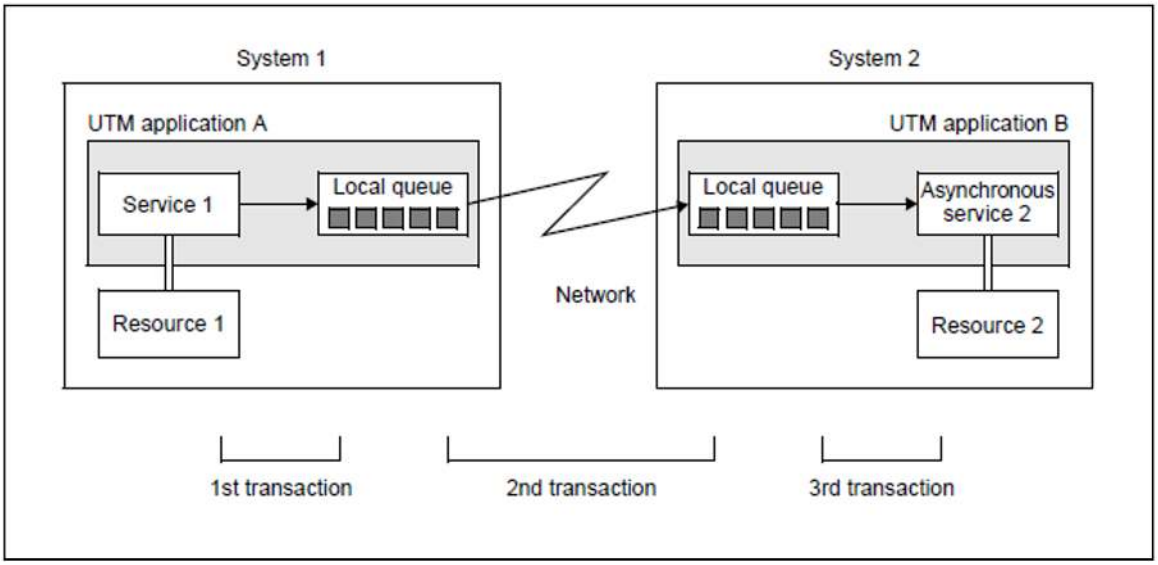


Figure 24: Remote queuing with openUTM

In figure 24, you can see how openUTM handles a background job sent to a remote application. This diagram also illustrates how remote queuing is often a practical alternative to distributed dialog processing. Communication between the two services is divided into three transaction-oriented steps. As soon as the job is entered in the local queue of application A, the required resources can be released in application A, and any locks set in resource 1 can be canceled - even if the network is currently unavailable or application B is not running.

If this form of communication were to be implemented within a dialog transaction, you run the risk of incessant locks and “hanging” transactions. By using remote queuing, however, it is possible to avoid ongoing global blockades caused by local errors or crashes.

5.1.3 Priority scheduling of background jobs

Background jobs are often used to off-load particularly time-consuming tasks. In this case, you must ensure that the background jobs do not occupy too many application processes, since this may affect the response times for dialog processing.

openUTM therefore offers a two-step priority scheduling concept:

- It is possible to define the number of processes of a UTM application which can simultaneously execute background jobs. This ensures that there is always a sufficient number of processes available for dialog jobs.
- It is also possible to differentiate between different background jobs. The background services are combined in TAC classes. There are two alternatives here:
  - Limiting the number of processes:
    The maximum number of processes that can be utilized at one time for a TAC class can be specified for each TAC class.
  - Priority control:
    Jobs from a TAC class with higher priority are processed first. You can choose between absolute, relative and equal priorities. “Absolute” means that *all* jobs with a higher priority TAC class must be processed before the next TAC class is processed. Relative priority means that TAC classes with higher priorities will be processed more often than TAC classes with lower priorities.

  You may only choose one of these two alternatives within an application.

You can find out how to specify the maximum number of processes that are available for background jobs or how to define and prioritize TAC classes in the openUTM manual “Generating Applications” under the keywords ASYNTASKS, TACCLASS and TAC-PRIORITIES.

5.2 Service-controlled queues

The messages in a service-controlled queue must be retrieved by a service of the application before they can be processed. In other words, the initiative must always come from a service of the application, since openUTM does not take on any triggering functionality for these queues.

openUTM provides three types of service-controlled queues:

- USER queues:
  A USER queue is user-specific and available automatically to every UTM user.
- TAC queues:
  These queues are available in principle to every service. They have a fixed name that has to be generated explicitly. An exception is the dead letter queue, which is permanently named KDCDLETQ and which does not have to be generated.
- Temporary queues:
  A temporary queue is generated and deleted again dynamically by a service by means of a program call.

openUTM supports both the browse and the processing function for all service-controlled queues:

- Browse
  The message can be read by several services in parallel and remains in the queue after it has been read.
- Process
  The message can be read only by one service at a time and is deleted after it has been read.

Both service-controlled queues and UTM-controlled queues are subject to the openUTM transaction concept.

- The messages are queued with transaction management and remain there, fault-tolerant, until they are retrieved with transaction management by the services and further processed.
- If a transaction is rolled back, the messages processed in the transaction are placed back in the queue and can be read again (redelivery). Alternatively, or after the last redelivery attempt, openUTM can put the incorrectly processed FGET message of a TAC queue in the dead letter queue.

You can set the maximum number of redeliveries after a transaction rollback. Details are available in the openUTM manual “Generating Applications”, keyword REDELIVERY.

5.2.1 USER queues

USER queues are permanent service-controlled message queues. A USER queue is available to every generated UTM user at all times. Any service can access USER queues by means of a program call provided it knows the name of the user. To prevent uncontrolled access, USER queues are integrated in the openUTM authorization concept. This allows you to assign the access rights role-specifically.

A USER queue exists for as long as the associated UTM user ID.

You can use USER queues to implement mailbox applications for UTM users, for example. The example in the figure below shows a further application of USER queues, in which the queues are used for the purpose of sending messages or warnings asynchronously to UPIC clients.

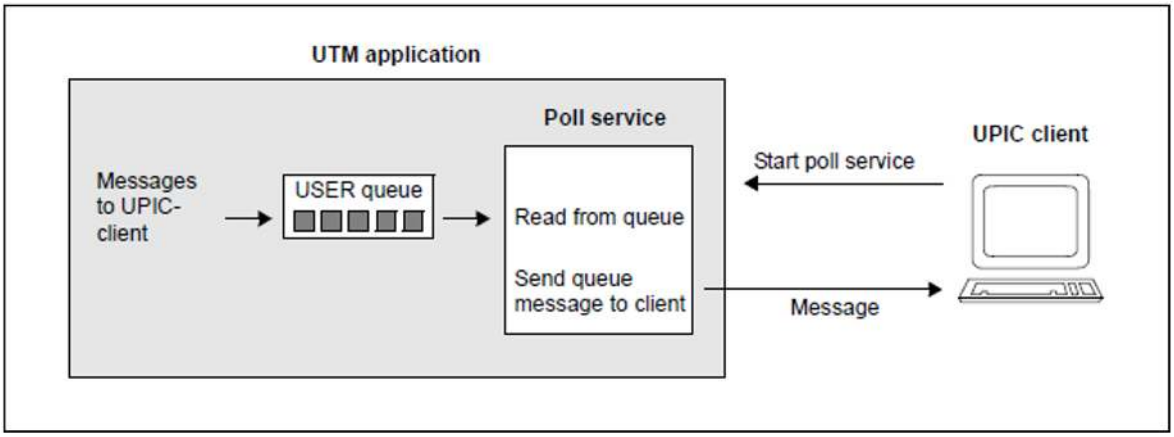


Figure 25: Example: Use of a USER queue for asynchronous messages to a UPIC client

In this example, use is made of the fact that a UPIC client signs on with openUTM using a UTM user ID. Asynchronous messages to the UPIC client go via the USER queue of this user ID.

The poll service is started by the UPIC client at the beginning of the dialog. It reads the message from the USER queue of the user ID and sends it immediately to the UPIC client as a dialog message. If there is no message in the queue, the poll service waits until a message is received. If the UPIC client program is programmed with multi-threading, the poll service and the normal dialog can run concurrently. In this way, the user on the client can automatically be kept informed of the arrival of asynchronous messages (e.g. by means of an icon or a dialog box).

To find out how to define USER queues and specify their properties, refer to the openUTM manual “Generating Applications” (look for the keywords USER, Q-READ-ACL, Q-WRITE-ACL and USER queue).

5.2.2 TAC queues

TAC queues are permanent, service-controlled message queues. They have a fixed name that is specified at generation. Any service can access TAC queues by means of a program call, provided it knows the name of the queue. To prevent uncontrolled access, TAC queues are integrated in the openUTM authorization concept. This allows you to assign the access rights role-specifically.

A TAC queue exists for an unlimited period of time unless it is explicitly deleted during administration.

Example for the usage of TAC queues

The diagram below shows an example of how you can use TAC queues to forward messages to the WinAdmin graphical administration workstation and archive them there. This example can be carried out analogously using WebAdmin.

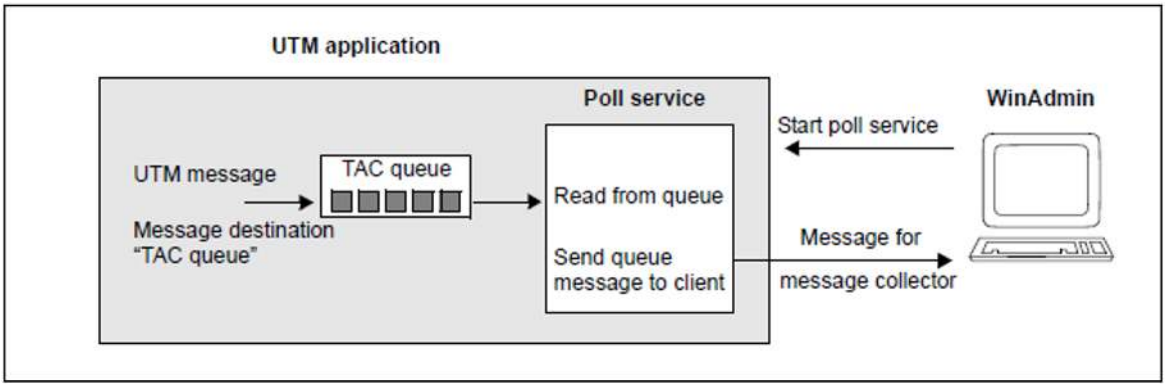


Figure 26: Example: Use of a TAC queue for WinAdmin message collectors

The message collector is defined in WinAdmin. It represents a polling mechanism that retrieves the UTM messages from the TAC queue either cyclically or when explicitly requested, depending on the setting in WinAdmin. On the UTM side, all that needs to be done in this case is for the TAC queue to be generated and a message module to be created in which it is specified which messages are to be sent to the TAC queue.

The poll service for WinAdmin and WebAdmin is shipped with openUTM (KDCWADM program unit).

To find out how to define TAC queues and specify their properties, refer to the openUTM manual “Generating Applications” (look for the keywords TAC, Q-READ-ACL, Q-WRITE-ACL and TAC queue).

5.2.3 Temporary queues

Temporary queues are created by means of a program call and can also be deleted again by means of a program. The name of the queue is assigned when the queue is created (i.e. it does not have to be generated). The name can be created by the program or by openUTM.

In the case of UTM-S, temporary queues remain after an application is terminated unless they are deleted explicitly beforehand. In the case of UTM-F, all temporary queues are lost when the application is terminated.

Temporary queues are particularly suitable for free communication between mutually independent services (“free” dialog), and they embody the concept of **synchronous waiting for asynchronous events** particularly well. They are thus considered to be at some point between the concepts of the “strict dialog” and the “background jobs”.

The services can either reside within an application or be located on different systems. The following are two important applications of this concept:

- the dialog between two services in a UTM application
- the dialog of a UTM application with a remote transport system application

These two applications are explained below and illustrated graphically.

In the first example, the “Customer” service of a call center application looks for additional customer data by means of the asynchronous service “Search”. This writes the data to a temporary queue, which is used in this case as a pure reply queue (i.e. data transfer goes in one direction only).

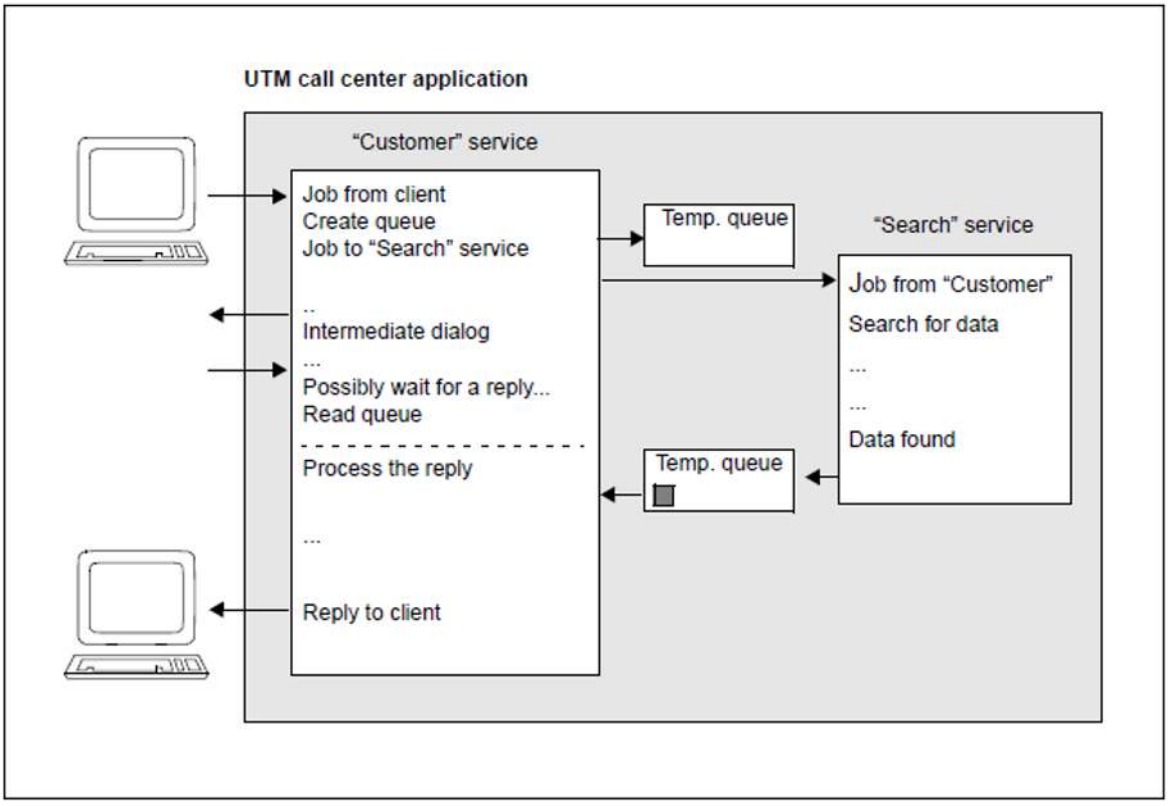


Figure 27: Communication between two services by means of a temporary queue

The “Customer” service creates the temporary queue and forwards the name of the queue in the job to the “Search” service. The “Customer” service then engages in further dialog queries with the client, while the “Search” service carries out a database query in the background. If necessary, the “Customer” service waits for the reply from the “Search” service (synchronous waiting). When it receives the required data, it writes it to the queue and then terminates. As soon as the message is in the queue, “Customer” is activated, takes over subsequent processing and then deletes the queue.

In the second example, a UTM application wants to access the data of a remote transport system application. As a pure transport system application, this cannot use any higher-level protocols such as OSI TP or LU6.1 for communication. In order to enable a dialog in spite of this, temporary queues are used on the UTM side.

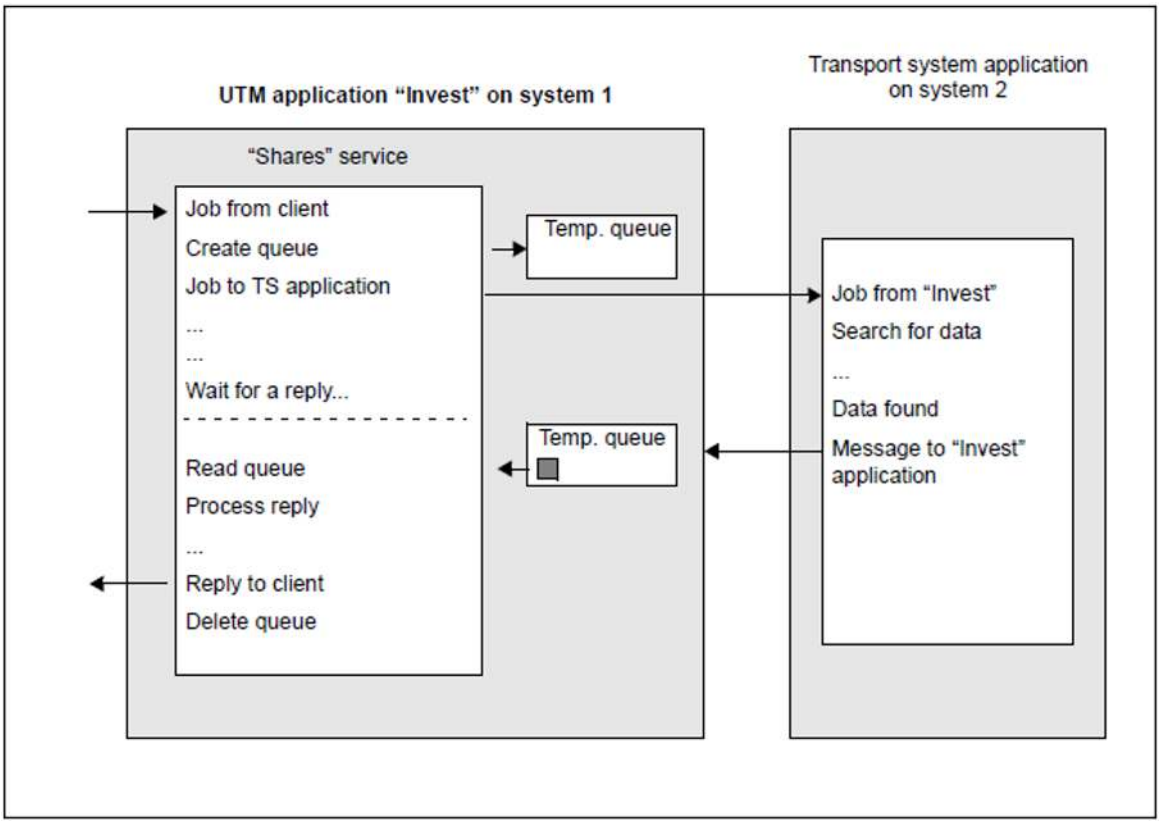


Figure 28: Communication with a transport system application by means of a temporary queue

The “Shares” service creates the temporary queue and forwards the name of the queue to the transport system in the job. The “Shares” service then waits for the reply (synchronous waiting). In the meantime, the transport system application obtains the required data and sends it to the “Invest” application. The name of the queue must be at the beginning of the message. The “Shares” service is activated as soon as the message is in the queue. The queue is deleted on completion of processing.

The openUTM manual „Programming Applications with KDCS” describes how to create and delete temporary queues. To find out how to specify the maximum number and properties of temporary queues, refer to the openUTM manual “Generating Applications” (look for the keywords QUEUE and temporary queue).

5.3 Control options for message queues

openUTM offers various options for UTM-controlled queues and service-controlled queues. These options are integrated in the MQ calls of the KDCS interface.

Confirmation jobs

In addition to the message to the queue (“basic job”), you can define up to two **confirmation jobs** which are linked to the success or failure of the basic job. Confirmation jobs are possible for the following types of basic jobs:

- background jobs
- output jobs to terminals, printers or transport system applications
- jobs to TAC queues

Confirmation jobs are initiated as soon as processing of the basic job is complete. They allow the job submitter to respond to a positive or negative job result. The confirmation job that does not apply, e.g. the negative confirmation job in the case of a successful basic job, is simply ignored. The basic job and the confirmation job are combined to form the **job complex**. What happens depends on what type of queue is involved:

Local background job

The positive confirmation job is started once the asynchronous service was terminated normally.

The negative confirmation job is started if the asynchronous service was terminated abnormally and if redelivery is not activated or the maximum number of redeliveries has been reached (see "Redelivery"). Abnormal termination may be due to a UTM call in the program (PEND ER/FR) or due to a major program error.

Background job to a remote service (remote queuing)

The positive confirmation job is started as soon as the job has been transferred successfully to the remote service queue.

The negative confirmation job is executed if the transfer fails despite repeated attempts.

Output job

The positive confirmation job is started once the positive print acknowledgment has arrived from the printer or printer administration.

The negative confirmation job is started:

- after errors during message editing by VTSU-B or errors during formatting,
- if the job is deleted during connection setup/shutdown by clients generated with RESTART=NO,
- if a job was deleted by administration.

However, the negative confirmation job is not started for a negative print acknowledgment, for a timeout when waiting for the acknowledgment, or if the print job is repeated.

Job to a TAC queue

The positive confirmation job is started once the message has been read successfully in the TAC queue and the transaction has been terminated normally.

The negative confirmation job is started:

- if the message is deleted with DADM DL and it is explicitly stated that the negative confirmation job is to be started (KCMOD=N),
- if the transaction was rolled back during message processing and redelivery is not activated or the maximum number of redeliveries has been reached, see section “Redelivery”.

Time control

It is possible to delay execution of background jobs, output jobs to LTERM partners and LPAP partners and jobs to TAC queues until a specified time. You can specify when the job is to be executed at the earliest or when the message can be read at the earliest. This can be defined relative to the time at which the job was submitted, or as an absolute value. This type of job is known as a **time-driven asynchronous job**. After the specified point in time, processing of the time-driven asynchronous job is started by openUTM as soon as the resources required to execute the job are available.

Redelivery

Redelivery of messages can be controlled for background jobs and messages to servicecontrolled queues. The following can be set by means of generation:

- whether a message to an asynchronous service is redelivered after abnormal service end,
- whether a message to a service-controlled queue is redelivered after a transaction has been rolled back,
- the maximum number of redeliveries. Different values are possible for background jobs and service-controlled queues. Specifying an upper limit prevents, for example, endless loops.

The redelivery function ensures that messages are preserved after service termination/transaction rollback and are not immediately deleted. openUTM also makes available a redelivery counter that can be read by the program unit. This allows the program to detect “loop situations” and to respond appropriately.

If a message is redelivered, no negative confirmation jobs are activated.

Backing up incorrectly processed jobs in the dead letter queue

The dead letter queue is a TAC queue that is always named KDCDLETQ. The dead letter queue is always available and is used to save messages sent to transaction codes, TAC queues, LPAP or OSI-LPAP partners that could not be processed or delivered.

In the generation, the backing up of messages in the dead letter queue can be enabled or disabled for each message destination.

For asynchronous TACs and TAC queues, the backing up can be done as an alternative to redelivery or as a last fallback stage once the maximum number of redelivery attempts has been reached.

To have the option of processing the messages in the dead letter queue even after the errors have been resolved, you can assign the messages either to their original destination or to a new target. Note that when you assign a new destination, the messages can be moved only to a destination of the same type as the original destination, i.e., either TAC, LPAP, or OSI-LPAP partner.

The number of messages in the dead letter queue can be monitored with message K134.

> The dead letter queue is stored in the page pool (see "KDCFILE for a standalone application"). If the dead letter queue is used, the page pool should be generated with sufficient space.

Information about activating and deactivating the backup of messages in the dead letter queue and monitoring the message volume can be found in the openUTM manual “Generating Applications” (parameter DEAD-LETTER-Q of the TAC, LPAP, and OSI-LPAP statement or parameter DEAD-LETTER-Q-ALARM in the MAX statement).

Information about moving or deleting messages from the dead letter queue or avoiding a page pool bottleneck can be found in the openUTM manual „Programming Applications with KDCS”, see KDCS call DADM and “Page Pool Bottleneck“).

Administration of message queues

The processing of messages and jobs in message queues can be influenced by means of UTM administration. This applies both to UTM-controlled queues and to service-controlled queues. An asynchronous job or a message in a USER queue, can, for example, be brought forward in the processing order or canceled. The KDCS call DADM is available for the administration of message queues (see the next section). Alternatively you can use the graphical administration workstation WinAdmin or WebAdmin for this purpose.

5.4 Message queue calls of the KDCS interface

openUTM offers convenient but powerful calls for message queuing functions at the program interface.The prefix “free” in the call names is intended to convey the fact that message queuing involves a form of communication which is independent of the sender and of the availability of the receiver.

- FPUT (**F**ree message **PUT**)

  FPUT calls are used to send asynchronous messages to output devices (output jobs), asynchronous services (background jobs) or TAQ queues. An asynchronous message can consist of several message parts, each of which requires a separate FPUT call.

- DPUT (**D**elayed free message **PUT**)

  DPUT calls are also used to send asynchronous messages or message parts to output devices or to asynchronous services. Unlike FPUT calls, however, DPUT calls offer the time control functionality and the option of using confirmation jobs. In addition, DPUT calls can also be used to address USER queues, TAC queues or temporary queues.

- FGET (**F**ree message **GET**)

  FGET calls are used to read asynchronous messages or message parts within an asynchronous service.

- DGET (**D**ata **GET**)

  DGET calls are used to read messages from USER queues, TAC queues or temporary queues.

- QCRE (**Q**ueue **CRE**ate)

  QCRE calls are used to create temporary message queues dynamically.

- QREL (**Q**ueue **REL**ease)

  QREL calls are used to delete temporary message queues dynamically.

- MCOM (**M**essage **COM**plex)

  MCOM calls are used to assign confirmation jobs to an asynchronous job.

- DADM (**D**elayed free message **ADM**inistration)

  DADM calls are used to request general information on the contents of a queue or on individual elements. They can also be used to control the queue sequence: jobs can be brought forward or cancelled, or all the jobs in the entire queue can be deleted.

The precise format of the FPUT, DPUT, FGET, DGET, MCOM, QCRE, QREL and DADM calls and additional information on these calls can be found in the openUTM manual „Programming Applications with KDCS”. The DADM call is described in the openUTM manual “Administering Applications”.

6 Structure of a UTM application

The purpose of a UTM application is to provide services: it processes the service requests (jobs) received from terminal users, client programs, or other applications.

Several UTM applications can exist in parallel in a single system. These applications are completely independent of each other, and can be generated and administered individually. If desired, it is possible to implement suitably organized job complexes in separate applications.

A UTM application consists of a UTM application program which is started in a certain number of processes defined by the user, the KDCFILE which is used by all processes as the “system memory”, and a UTM cache memory which optimizes access to the KDCFILE.

From a technical point of view, a UTM application is a group of processes which forms a logical server unit at runtime.

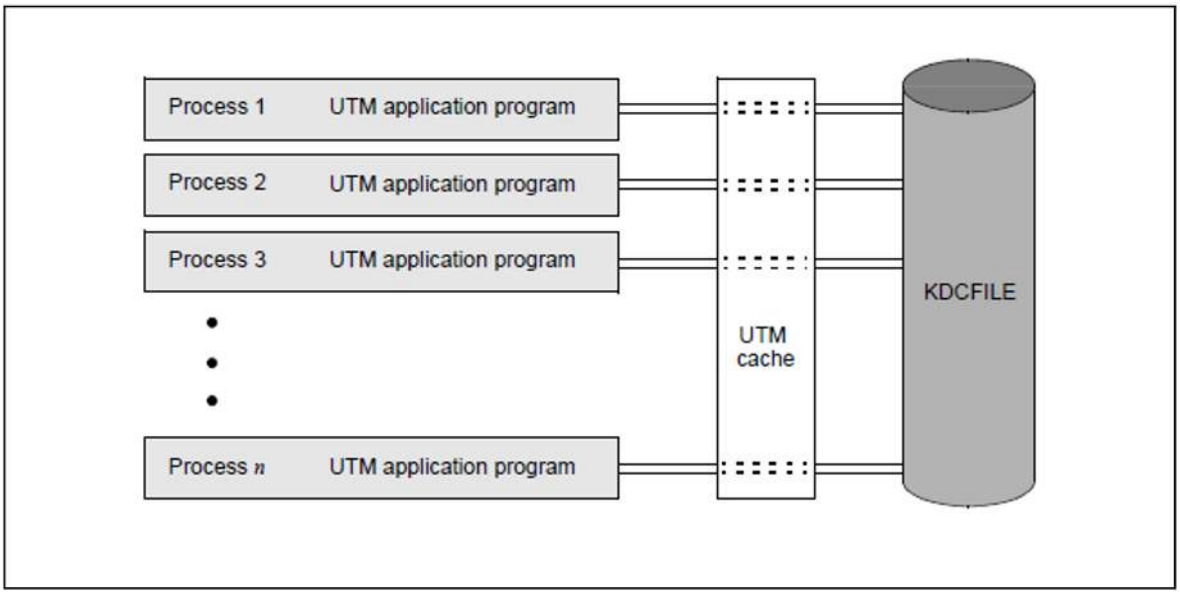


Figure 29: Structure of a UTM application

6.1 UTM application program

When designing an application program, you must define **service routines** (also known as **program units**) in which you define the services to be provided by your application. These service routines can be programmed in one of the common programming languages (C/C++/COBOL).

Through the integration of **UTM calls**, the service routines access the UTM system functions, e.g. for transaction management, sending and receiving messages etc. (see also chapter “Program interfaces”).

The service routines are assigned **transaction codes** (TACs) either during generation using the KDCDEF statement TAC, or during operation using the KC_CREATE_OBJECT call for the object type KC_TAC. The transaction codes are freely selectable names used by terminal users, clients, or other programs to start the service routines.

To ensure that the service routines can run under the management of openUTM, the compiled service routines are linked to the **UTM application program** (see "Generating the application program") together with other modules (allocation tables, messages, libraries used, etc.). As an alternative to the static linking of the service routines, these can be dynamically linked when an application process is started or the first time a service is called.

One part of the application program is the **main routine KDCROOT**, which acts as the main control program and is responsible for coordinating job sequences.

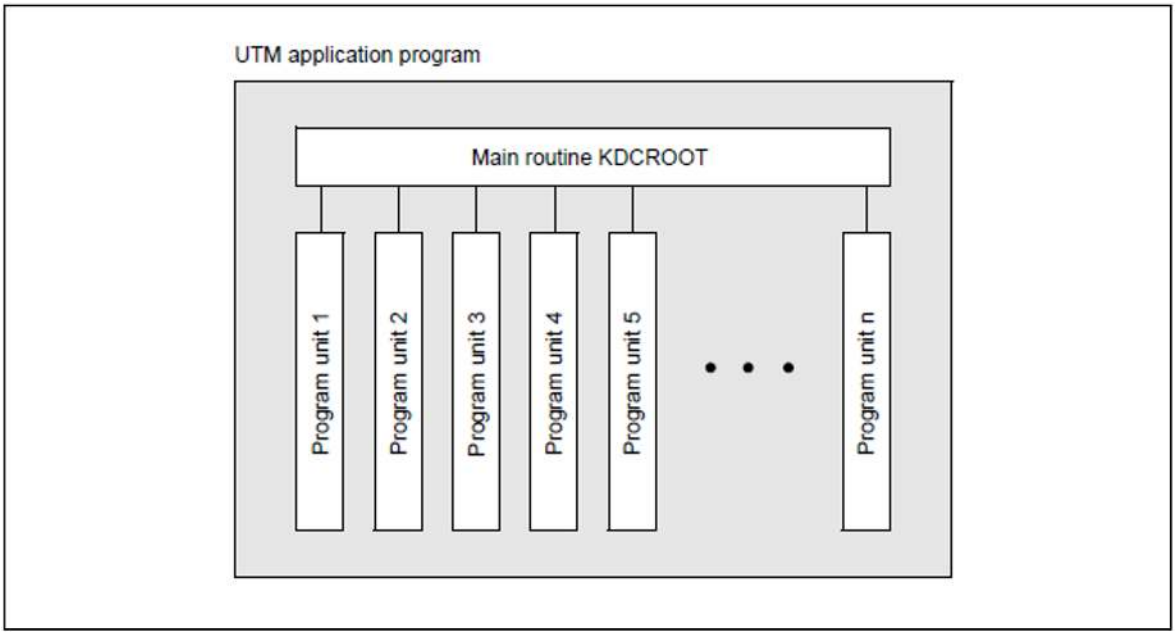


Figure 30: Structure of a UTM application program

6.2 The process concept

When the application is started, the application program is initiated in a certain number of processes defined by the user.

Since a UTM application is generally accessed simultaneously by a number of clients, each client does not have its own exclusive process. Instead, the large number of simultaneous requests is distributed by openUTM across a small number of processes. As a result, the system overhead and thus the response times do not increase proportionally with the number of users.

If the number of jobs to be processed exceeds the number of free processes, openUTM places the jobs in a queue. Similarly, if the number of free processes exceeds the number of outstanding jobs, the free processes are placed in a process queue.

In addition to the number of processes defined by you, further processes are also started for UTM applications. These are known as **UTM system processes**, see section "UTM system processes" below.

This openUTM concept of several homogeneous peer processes has a number of further advantages:

- Several jobs can be processed simultaneously.
- If several jobs request the same service, the service can be provided simultaneously in different processes.
- A UTM application can react quickly to load fluctuations, since processes can be added or removed during operation by administration.
- Throughput bottlenecks in particular services can be eliminated, since the processes are of equal ranking (homogeneous) and can be used for any job.
- The worst possible outcome of a fatal error in an application program is a process crash. The effect of the error thus remains local, i.e. only the job that was being processed by the process at the time of the crash is affected. The application as a whole and other applications running in the same system under the control of openUTM are not affected. openUTM will automatically replace the aborted process by a new one.

UTM system processes

The purpose of the system processes is to ensure that applications continue to be responsive even under high loads. UTM system processes only process selected jobs which are characterized first and foremost by short runtimes. Selected jobs may be, for example, the establishment or cleardown of connections, timeout displays, end-of-transaction requests in UTM-D and responses for programs waiting for a PGWT call.

Selected jobs may also include program unit runs for an administrator. On generation, it is also possible to specify a so-called privileged LTERM partner. The following applies to the connection associated with this LTERM partner:

- If a sign-on service is started for this connection then this sign-on service is also processed by the UTM system processes.
- If an administrator signs on via this connection then program unit runs for this connection are also handled by the UTM system processes.
- If a normal user signs on via this connection then this connection is handled exclusively using "normal" processes until the user signs off.

UTM starts the UTM system processes implicitly when an application is started. The number of UTM system processes that are started depends on the values specified in the start parameter TASKS and in the generation parameter MAX SYSTEM-TASKS.

The following table shows the number of UTM system processes in case of MAX SYSTEM-TASKS=*STD has been specified in the UTM generation.

Start parameter TASKS= (MAX SYSTEM-TASKS=*STD)	Number of additionally started UTM system processes	Total started processes
1	0	1
2	1	3
3	2	5
4	2	6
5	3	8
n > 5	3	n + 3

Optimum process utilization through pseudo-conversations

Service routines can be programmed such that a user does not occupy a process during wait times, e.g. while he /she is thinking about what to do. The process then is released immediately, and is available for other jobs. When the terminal user has finished inputting information, another process may continue the dialog without the user noticing any difference. openUTM thus guarantees the optimum utilization of processes, which has a positive effect on performance.

This dialog concept, also known as pseudo-conversational, can be applied not only for dialogs with terminal users but also for program-to-program communication.

Further information on the subject of pseudo-conversations can be found in the openUTM manual “Programming Applications with KDCS for COBOL, C and C++”.

Some of the operating-system-specific aspects of the process concept are dealt with in chapters “openUTM on BS2000 systems”, “openUTM on Unix and Linux systems” and “openUTM on Windows systems”.

6.3 The KDCFILE - the “application memory”

The content and role of a KDCFILE differ depending on whether the associated application is a standalone application (see below) or a UTM cluster application (see "KDCFILEs in UTM cluster applications on Linux, Unix and Windows systems").

6.3.1 KDCFILE for a standalone application

The KDCFILE of a standalone application consists of one or more files and contains the data required to run a UTM application. It is created using the KDCDEF generation tool (see "Defining the configuration"). When operating a UTM application, the KDCFILE is shared by all processes within the application.

The KDCFILE is logically divided into the following three areas:

- administrative data
- the page pool
- the restart area

For data protection, the KDCFILE can be mirrored on different disk drives.

To avoid disk bottlenecks and improve access times, the page pool and the restart area can be distributed across several files.

A detailed description of the KDCFILE can be found in the openUTM manual “Generating Applications”.

Administrative data

The administrative data area contains information on the configuration, e.g. parameters of the UTM application, lists of the contents of all objects that can be addressed by name, administrative information on the page pool and the restart area, as well as tables of services, user IDs, clients, transaction codes, lock codes, and function keys.

When a UTM application is started, the administrative data used by the application processes, and about which they communicate, is provided in a shared memory area. Any changes to the administrative data are written back to the KDCFILE while the application is running and after the application is terminated. This ensures that the data is retained for the next application run, and allows for an automatic restart if the UTM application is terminated abnormally.

Page pool

The page pool contains all user data that arises during a UTM application run. Examples include:

- various secondary storage areas
- information for a screen restart
- communication areas
- queues containing output jobs (including time-driven output jobs) and background jobs
- messages of service-controlled queues
- buffered records from the user log file

The active UTM application can access the page pool via the UTM cache. The size of the page pool (number of UTM pages) is defined during generation.

Restart area

The UTM calls in a program unit result in changes to the administrative data and user data. openUTM collects information on all modifications that occur within a transaction.

In the case of a UTM-S application (see "Restart with UTM-S"), openUTM uses this information to generate a data record containing restart information at the end of the transaction. It then writes this data record in the restart area of the KDCFILE. The data record describes the changes that must be made to the administrative data as a result of the transaction. The size of the restart area determines how often modifications to the administrative data should be transferred to the administrative data area of the KDCFILE. You can use an administration command to find out what this interval is.

In the case of a UTM-F application (see "Restart with UTM-F"), restart data records are written only for transactions in which passwords were changed or administrative data was modified by means of dynamic configuration.

6.3.2 KDCFILEs in UTM cluster applications on Linux, Unix and Windows systems

Every node application in a UTM cluster application has its own KDCFILE. Because some date applies locally at node level while other data is global to the cluster there are some differences compared to standalone applications:

- Data that applies locally to the node is stored only in the KDCFILE of the relevant node application. Data that applies locally at node level includes, for example, TLS, asynchronous messages, background jobs and data relating to other node-bound services. The KDCFILEs of the individual node applications are not identical at runtime.
- Data that applies globally to the cluster is stored in UTM cluster files, see section “UTM cluster files”. This includes user data such as GSSB, ULS, users’ service data and passwords.'

The "memory" of a UTM cluster application therefore consists of the sum of all the local KDCFILEs and the UTM cluster files.

Please note that it is not possible to keep duplicate KDCFILEs in a UTM cluster application, i.e. there is only ever one KDCFILE for a node application.

6.4 UTM cache memory

The local UTM cache memory is an area in virtual memory which is managed in 2KB, 4KB or 8KB units, depending on the generation. openUTM uses this cache as a global buffer area for access to the page pool, i.e. the processes of a UTM application handle all access to page pool data via this storage area. If the cache memory is sufficiently large, UTM allows you to optimize read/write access to the page pool. The main advantage of this is that read operations can be reduced if page pool data from a previous transaction (possibly written by another process) is still available in the cache memory.

> **i** In UTM cluster applications, the cache memory is also used as buffer area for accesses to the cluster page pool.

The following values are defined during UTM generation:

- the size of the cache memory
- the percentage of cache memory pages to be written to the page pool if space is required for new data; this can be modified during operation by administration
- whether the cache is to be located in the program address space or in one or more data spaces (only on BS2000 systems)

The administration functions allow you to find out whether the cache memory is being used in a currently running application.

There is no universal formula for determining the ideal value for this parameter. You will have to base your decision on comparisons and performance measurements. However, you will find a few pointers and tips in the openUTM manual “Generating Applications”.

7 Program interfaces

The openUTM program interfaces allow for fast, simple programming of the most varied forms of communication for a wide range of scenarios.

openUTM supports the KDCS interface (German standard), as well as the CPI-C, XATMI and TX interfaces standardized by X/Open, and the interfaces UTM-XML and UTM-HTTP:

- KDCS is a universal program interface for transaction-oriented applications and is standardizes according to DIN 66 265
- CPI-C and XATMI are interfaces for program to program communication
- TX is a program interface used to define transactions
- UTM-XML is an interface used to edit XML documents
- UTM-HTTP is an interface that provides additional functions for communication with HTTP clients.

7.1 Overview of the program interfaces

This section describes which program interfaces are available in which programming languages for servers and for clients and how they can be combined with each other.

Program interfaces for openUTM servers

The following table provides an overview of the language environments in which the four program interfaces are available on the server platforms.

Server platform	**Language environments available for the interfaces**					
	KDCS	**UTM-HTTP**	**CPI-C (X/Open)**	**XATMI (X/Open)**	**TX (X/Open)**	**UTM-XML**
BS2000 systems	COBOL, C, C++, Assembler, Fortran, PL/I, Pascal-XT	C, C++	C, C++ COBOL	C, C++ COBOL	C, C++ COBOL	C, C++ COBOL
Unix, Linux, Windows systems,	COBOL, C, C++	C, C++	C, C++ COBOL	C, C++ COBOL	C, C++ COBOL	C, C++ COBOL

TX is a program interface used to define transactions. It is always used together with the CPI-C (explicitly) or XATMI (implicitly) communication interface and cannot be used alone.

A combination with KDCS and TX is not allowed.

The UTM-HTTP interface can only be used in combination with KDCS.

It makes sense to use the following combinations in an application using UTM-XML:

- KDCS + UTM-XML
- CPI-C + TX + UTM-XML
- XATMI + UTM-XML

A combination with KDCS and TX is not allowed.

Program interfaces for UTM clients

The UTM clients are available with the UPIC and OpenCPIC carrier systems. Both carrier systems offer a range of program interfaces.

The following table provides an overview of whether and in which language environments the various program interfaces are available on the carrier systems.

Client interface	**Language environment for UPIC**	**Language environment for OpenCPIC**
CPI-C (UPIC calls)	C, C++, COBOL	C, C++, COBOL
CPI-C (full scope of X/Open)	--	C, C++, COBOL
TX (X/Open)	--	C, C++, COBOL

XATMI (X/Open)	C, C++, COBOL	C, C++, COBOL
UTM-XML	C, C++, COBOL	C, C++, COBOL

Interface combinations for communication

For communication between client and server or between two servers, the following rules apply to the combination of interfaces:

- If none of the partners uses the XATMI interface, any combination of the interfaces in the tables can be used except for XATMI.
- Communication via the XATMI interface is only possible if **both** partners use the XATMI interface.
- Global transaction management is only possible when both partners use an interface with transaction management. For example:
  - KDCS with KDCS
  - KDCS with CPI-C + TX
  - XATMI with XATMI

7.2 The KDCS universal program interface

The KDCS interface was defined and standardized (DIN 66 265) as a non-proprietary interface for transaction-oriented applications.

It offers the following features:

- a comprehensive range of function calls which can be used universally(e.g. for pseudo-conversations, message queuing, or direct communication with terminals)
- KDCS-specific storage areas for simple, reliable programming
- event management functions

KDCS is available for C, C++ and COBOL, and on BS2000 systems for Assembler, Fortran,PL/I and Pascal-XT.

Detailed information on the KDCS program interface can be found in the openUTM manual “Programming Applications with KDCS for COBOL, C, and C++”. Separate supplementary manuals are also provided online for the additional programming languages supported on BS2000 systems.

7.2.1 KDCS calls

The UTM function calls can be divided into the following function groups:

- program and transaction management
- dialog communication
- communication via message queuing
- administration of message queues and printers
- storage area management
- information services
- logging
- signing on/off and changing passwords

Program and transaction management:

Call	Function
INIT	Sign on a program to openUTM
PEND	End the program
PGWT	Set a waiting point in a program unit run
RSET	Roll back requested changes and operations

Dialog communication:

Call	Function
APRO	Address a job-receiving service (for distributed processing)
CTRL	Control an OSI TP dialog (for distributed processing)
MGET	Get a dialog message
MPUT	Send a dialog message

Communication via message queuing:

Call	Function
APRO	Address a job-receiving service (for distributed processing)
DGET	Read a message from a service-controlled queue
DPUT	Create time-controlled asynchronous jobs, messages in service-controlled queues and confirmation jobs
FGET	Get asynchronous messages
FPUT	Create asynchronous jobs and messages in TAC Queues
MCOM	Define a job complex
QCRE	Create a temporary queue
QREL	Delete a temporary queue

Administration of message queues and printers:

Call	Function
DADM	Administer messages in UTM-controlled and service-controlled queues and administer the dead letter queue
PADM	Control printers and print jobs

Storage area management:

Call	Function
GTDA	Read from a TLS
PTDA	Write to a TLS
SGET	Read from a secondary storage area
SPUT	Write to a secondary storage area
SREL	Release a secondary storage area
UNLK	Unlock a TLS, ULS, or GSSB

Information services:

Call	Function
INFO	Request information

Logging:

Call	Function
LPUT	Write to the log file

Signing on/off and changing passwords:

Call	Function
SIGN	Sign on/off, change passwords

Return codes of KDCS calls

After each KDCS call, openUTM returns a standard KDCS return code and, if necessary, a UTM-specific return code.

Among other things, these return codes indicate whether or not the desired operation was successful. The return codes are divided into the following categories:

- no errors
- minor errors that can be rectified by the program
- codes for function keys that were pressed while an input message was being transmitted
- errors that still allow you to end the dialog step or the service
- serious errors that cause openUTM to roll back the transaction and terminate the service with a dump. In this case the return code can be taken from the dump. If possible, openUTM sends an error message to the communication partner.

An overview of all return codes can be found in the openUTM manual ”Messages, Debugging and Diagnostics”.

7.2.2 UTM storage areas

openUTM offers program units for reading and writing user data in various storage areas. These storage areas provide a clear distinction between program and data areas, as well as guaranteeing the reentrant capability of programs. They also allow for high-performance, transaction-oriented communication between programs, and ensure effective memory utilization. Some storage areas are designed specifically for statistical or logging purposes.

> In this sense, service-controlled queues can also be considered as UTM storage areas. Since service-controlled queues are designed specifically for message queuing, they are described separately in the chapter “Message queuing” especially in subchapter "Service-controlled queues" .

Primary storage areas

These are storage areas which are available to the program units in the main memory:

- Communication area (KB)

  openUTM creates the communication area when a new service is started. The contents of the communication area are transferred to the program unit currently being executed. In the communication area, openUTM provides the program units with up-to-date information. This area is also used for communication between the program units of a service. Its size can be adapted to the data to be transferred. The communication area is subject to transaction security and continues to exist until the service is terminated.

- Standard primary working area (SPAB)

  By default, openUTM assigns a standard primary working area to each program unit. This area is then available to the program unit from the start of the program to the end of the program (PEND call). The data in this area therefore cannot be stored or forwarded once the program unit run is terminated. The SPAB can be used for the parameter area in which the program unit supplies the parameters for KDCS calls. It can also be used to buffer messages, for example. Since the standard primary working area is reserved for a specific program unit, it is **not** integrated in transaction management. One advantage the SPAB has over other forms of program memory (e.g. stacks) is that the SPAB is output in the UTM dump.

- Other areas

  A program unit can use other areas for storing data. These areas are defined during generation using the KDCDEF statement AREA and must be managed by the users themselves, they are **not** subject to transaction management. Their structure is not prescribed by openUTM and can be defined freely.

Secondary storage areas

These storage areas are implemented by openUTM in background memory that is subject to transaction management. They are read and written to using special KDCS calls:

- Local secondary storage area (LSSB)

  The local secondary storage area is a service-specific background storage area used to transfer data between program units within a particular service. While the communication area is automatically provided and logged for each program unit, the local secondary storage area is accessed only when required. It is therefore particularly suitable for read-only data, or for cases where several program units which do not require the respective data are to be executed between the write operation and the read operation. The local secondary storage area continues to exist until it is explicitly released or the service is terminated.

- Global secondary storage area (GSSB)

  The GSSB is a background storage area that allows for any data to be transferred between the services in a UTM application. Unless it is explicitly released, it continues to exist even after the application is terminated.
  In UTM cluster applications, GSSBs can be used cluster-wide; all node applications have the same view of the data of a GSSB.

- Terminal-specific long-term storage area (TLS)

  The TLS is assigned an access point (LTERM, LPAP or OSI-LPAP partner) and is used to store information that must be available irrespective of the duration of a service or the runtime of the application. For instance, it can be used to produce statistics for an LTERM partner.

  In UTM cluster applications, TLSs are only supported locally in the node.

- User-specific long-term storage area (ULS)

  A ULS can be assigned to each user ID during configuration. It is used to store information that must be available irrespective of the lifetime of the services or the runtime of the application. For instance, it can be used to produce statistics for each user ID.

  A ULS is also assigned a session (LU6.1) and an association (OSI TP).
  In UTM cluster applications, ULSs can be used cluster-wide; all node applications have the same view of the data of a ULS.

User log file (USLOG)

The user log file is a log file managed by openUTM in which user-specific information can be written using the KDCS call LPUT. It is implemented in the form of a file generation group (see the openUTM manual “Generating Applications”). The user log file can be used to record attempts to violate the data access control mechanisms, for example.

Each node application has its own user log file in UTM cluster applications.

The user log file is subject to transaction management and is therefore **not** suitable for use as a general log function. This is because the LPUT information is discarded when a transaction is rolled back.

Overview: UTM storage areas

Abbreviation	Type of Area	Life Cycle	Function	Transactional Backup
KB	Communication area	From the start of the service until the end of the service	Access to up-to-date information provided by openUTM; communication between the program units of a service	Yes
SPAB	Standard primary working area	From the start of the program unit until the end of the program unit	Transfer of parameters for KDCS calls; message buffer	No
AREA	Other storage area defined during generation	For the duration of the application	Store global application data, preferably for read-only purposes; In UTM cluster applications, AREAs only apply locally to the node.	No
LSSB	Local secondary storage area	From the first write call until explicit release or until the end of the service	Data exchange between the program units of a service	Yes
GSSB	Global secondary storage area	From the first write call until explicit release or until deletion of the application information	Distributed data exchange	Yes
ULS	User-specific long-term storage area	From generation until the generation is modified	For example, statistics for particular user IDs, sessions or associations	Yes
TLS	Terminal-specific long-term storage area	From generation until the generation is modified	Statistics for particular access points (LTERMs, LPAPs, OSI-LPAPs)	Yes
USLOG	User log file	Defined individually	Logging	Yes

7.2.3 Event functions

To enable you to react systematically to certain standard situations, openUTM offers the option of using event functions. Unlike “normal” program units, which are called by specifying a transaction code, openUTM automatically starts these program units when certain events occur.

There are two types of event function:

- event services, which **must** contain **KDCS calls**
- event exits, which **must not** contain any **KDCS calls**

The use of all the event functions is optional. You specify which event functions are to be activated in a UTM application during generation.

Event services

BADTACS	The BADTACS dialog service is started by openUTM when a terminal user or a transport system application specifies an invalid transaction code. For instance, BADTACS can be used to output help information or user prompting to inform the user of the transaction codes available for starting services.
MSGTAC	The MSGTAC asynchronous service is started by openUTM when UTM messages occur in the application, to which the MSGTAC message line has been assigned in the message file. The MSGTAC event service can be used to automate administration (see also "Automatic administration"). In the event of misuse, for example, the relevant terminal or a transaction code that results in repeated errors can be automatically locked.
SIGNON	The SIGNON dialog service is started by openUTM when a terminal user, a transport system application or a UPIC client signs on to the UTM application. A number of SIGNON services can be defined: one for each transport system access point. This means that services can be designed differently depending on the partner type. If a SIGNON service is generated for a transport system access point, it will always be executed for the terminals and transport system applications that establish the connection to the application via this access point. However, it is only activated for UPIC clients if it has been generated to do so explicitly.

The SIGNON dialog service allows you to customize the signon dialog for your application. For instance, in addition to the openUTM sign-on checks, you can carry out your own authorization checks (see also "System access control (identification and authentication)") or you can explicitly assign a UTM user ID to transport system applications in the sign-on service.

Event Exits

START — The START exit is called by openUTM when starting or reloading the application program in every process.

It can be used to open local files, for example.
Up to 8 different START exits can be defined.

SHUT — The SHUT exit is called by openUTM when terminating the application program in every process. It can be used to close local files, for example.

Up to 8 different SHUT exits can be defined.

VORGANG — The event exit VORGANG could be assigned to each individual service during configuration of a UTM application. It is then called by openUTM when starting and terminating the service - this applies even for abnormal terminations and service restarts. With this event exit, it is possible to access the KB header and the standard primary working area.

The Vorgang exit permits service-specific actions, e.g. the opening and closing of special resources for certain services.

INPUT — The INPUT exit is called each time an entry is made at the terminal. However, this does not apply for entries in the SIGNON event service.

The INPUT exit allows you to define which actions are to be initiated by input at the terminal, e.g. startup of a service or execution of a user command. It also offers a great deal of flexibility when designing the user interface.

HTTP — The HTTP event exit is used to appropriately reformat input messages from HTTP clients for the program unit and output messages from the program unit to the HTTP client.

The HTTP exit could be assigned to a TAC with the HTTP-DESCRIPTOR statement.

The FORMAT event exit is also available on BS2000 systems, and allows users to program their own formatting.

7.3 Program interface UTM-HTTP

openUTM provides a number of functions to read and write segments of an HTTP message from an HTTP exit or a program unit. These functions can be used to control the program flow and transfer segments of the response message to UTM.

The following list provides an overview. The functions are provided for the C / C++ programming language.

kcHttpGetHeaderByIndex	returns name and value of the HTTP header field with the specified index
kcHttpGetHeaderByName	returns the value of HTTP header field with the specified HTTP header name
kcHttpGetHeaderCount	returns the number of HTTP header fields of the HTTP request that can be read by the program
kcHttpGetMethod	returns the HTTP method of the HTTP request
kcHttpGetMputMsg	returns the MPUT message created by the program unit
kcHttpGetPath	returns the HTTP path of the HTTP request normalized
kcHttpGetQuery	returns the HTTP query of the HTTP request normalized
kcHttpGetRc2String	Help function to convert a function result of type enum into a printable null-terminated string
kcHttpGetReqMsgBody	returns the message body of the HTTP request
kcHttpGetScheme	returns the scheme of the HTTP request
kcHttpGetVersion	returns the HTTP version of the HTTP request
kcHttpPercentDecode	decodes the percent-encoded representation of a character
kcHttpPutMgetMsg	passes a message for the program unit which can be read with MGET
kcHttpPutStatus	passes an HTTP status code for the HTTP response
kcHttpPutHeader	passes an HTTP header field for the HTTP response
kcHttpPutRspMsgBody	passes a message for the message body of the HTTP response

You will find detailed information about the programming interface UTM-HTTP in the openUTM manual "Programming applications with KDCS".

7.4 The X/Open interface CPI-C

CPI-C (**C**ommon **P**rogramming **I**nterface for **C**ommunication) is a program interface for distributed program-to-program communication, which has been standardized by X/Open and the CIW (**C**PI-C **I**mplementor's **W**orkshop).

openUTM provides the CPI-C program interface for the programming languages COBOL, C, and C++. Under openUTM, CPI-C can communicate not only on the basis of the X/Open OSI TP protocol, but also using the LU6.1 and UPIC protocols.

Since CPI-C only supports program-to-program communication, it does not offer any functions for communicating with terminals. As a result, CPI-C services in openUTM cannot be started directly from a terminal (by entering a transaction code). The CPI-C services of a UTM application can only be started by means of service requests from other programs, such as:

- UTM-Client programs
- other UTM applications (server-to-server communication)
- remote applications (e.g. CICS applications in the case of server-to-server communication)

The following tables provide an overview of the CPI-C calls available in openUTM. The individual calls are described in detail in the X/Open CAE Specification on CPI-C (Version 2) of October 1994.

All UTM-specific details are described in the openUTM manual “Creating Applications with X/Open Interfaces”.

Overview: CPI-C calls in openUTM

(The call names are identical for C/C++ and COBOL.)

Calls from the starter set:

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

Calls for error and acknowledgment handling:

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

Calls for data conversion:

Function	Call	Description
Convert_Incoming	CMCNVI	Convert incoming data from EBCDIC into the character set used on the local system
Convert_Outgoing	CMCNVO	Convert outgoing data from the character set used on the local system into EBCDIC

7.5 The X/Open interface XATMI

XATMI is a program interface for distributed program-to-program communication, which has been standardized by X /Open.

openUTM provides the XATMI program interface for the programming languages COBOL, C and C++. Under openUTM, XATMI can communicate not only on the basis of the X/Open OSI TP protocol, but also using the LU6.1 and UPIC protocols.

XATMI services can only be started by partners that also use the XATMI interface.

XATMI differentiates between three communication models:

- Synchronous request/response model:
  After sending the service request, the client is blocked until a response is received.
- Asynchronous request/response model:
  The client is not blocked after sending the service request.
- Conversational model:
  The client and server can exchange data as desired.

The following tables provide an overview of the XATMI calls available in openUTM. The individual calls are described in detail in the X/Open CAE Specification on XATMI of November 1995.
All UTM-specific details are described in the openUTM manual “Creating Applications with X/Open Interfaces”.

Overview: XATMI calls in openUTM

Calls for the request/response model:

C/C++ call	Cobol call	Description
tpcall	TPCALL	Service request in the synchronous request/response model
tpacall	TPACALL	Service request in the asynchronous request/response model
tpgetrply	TPGETRPLY	Request response in the asynchronous request/response model
tpcancel	TPCANCEL	Cancel an asynchronous service request before a response has been received

Calls for the conversational model:

C/C++ call	COBOL call	Description
tpconnect	TPCONNECT	Set up a communication link
tpsend	TPSEND	Send a message
tprecv	TPRECV	Receive a message
tpdiscon	TPDISCON	Shut down a communication link

Call for the type-based buffers:

C/C++ call	COBOL call	Description
tpalloc	--	Reserve memory for a type-based buffer
tprealloc	--	Change the size of a type-based buffer
tpfree	--	Free a type-based buffer
tptypes	--	Query the type of a type-based buffer

General calls for service routines:

C/C++ call	COBOL call	Description
tpservice	TPSVCSTART	Provide a template for service routines
tpreturn	TPRETURN	Terminate a service routine
tpadvertise tpunadvertise	TPADVERTISE TPUNADVERTISE	Only supported syntactically in openUTM (output the name of a service routine)

7.6 The X/Open interface TX

TX (Transaction Demarcation) is a program interface for defining distributed transactions, which has been standardized by X/Open.

openUTM provides the TX program interface for the programming languages COBOL, C and C++.

With openUTM, TX calls should only be used in CPI-C services. XATMI services are included implicitly in global transactions by means of the TPTRANS or TPNOTRANS flag. The XATMI call *tpreturn()* determines whether a transaction is terminated successfully or rolled back.

With TX, transactions can be executed in chained or unchained mode. In chained mode, only the first transaction need be started explicitly: the end of this transaction implicitly marks the beginning of the next transaction.
In unchained mode, the beginning of each transaction must be specifically marked.

However, openUTM always works in chained mode. When a service is started under openUTM, a transaction is begun automatically. For this reason, the first transaction need not be marked.

The OpenCPIC carrier system also enables openUTM clients to control transactions with TX.

Overview: TX calls in openUTM

C/C++ call	COBOL call	Description
tx_commit	TXCOMMIT	Terminate global transaction successfully
tx_rollback	TXROLLBACK	Roll back global transaction
tx_info	TXINFORM	Query global transaction information
tx_set_commit_return	TXSETCOMMITRET	Set *commit_return* characteristic
tx_set_transaction_control	TXSETTRANCTL	Set *transaction_control* characteristic
tx_set_transaction_timeout	TXSETTIMEOUT	Set *transaction_timeout* characteristic
tx_open	TXOPEN	Always returns TX_OK under openUTM (open set of resource managers)
(The *tx_begin()* (TXBEGIN) and *tx_close()* (TXCLOSE) calls always return TX_PROTOCOL_ERROR under openUTM).		

The individual calls are described in detail in the X/Open CAE Specification on TX of April 1995.

All UTM-specific details are described in the openUTM manual “Creating Applications with X/Open Interfaces”.

7.7 The XML interface of openUTM

XML (Extensible Markup Language) is a logical description language that enables documents to be structured into elements. XML allows users to define their own language syntax, e.g. by means of a DTD (Document Type Definition) or an XML schema (XSD).

The XML interface of openUTM (short: UTM-XML) is a program interface with which XML documents can be created, processed and read. In addition, XML documents can be validated against XML schemas.

This interface is called inside of UTM program units and is available for the program languages C, C++ and COBOL.

Each of the interfaces described in this section (KDCS, CPI-C or XATMI) can be used as a communication interface. UTM-XML can also be used in openUTM clients.

A simple usage scenario such as processing an XML document in the KDCS program unit might look like this.

1. Read data with MGET

   The XML document is read into the message area of an XML schema (XSD).
2. Process the data via the UTM-XML interface

   The XML document is converted into an object tree and the individual elements are processed. Once processing is complete, the XML object is converted back into an XML document.

   A separate structure (t_value structure) is used for data type conversion.
3. Send the XML document with MPUT

   The processed XML document is made available in the message area and is sent back to the submitter.

If an XML document is processed interactively in several dialog steps (= program units), the programmer must ensure that the context is preserved. There are two ways of doing this.

- The XML object is converted into an XML document and buffered in a suitable UTM storage area, e.g. KB, LSSB or GSSB, see "UTM storage areas".
- The program uses the KDCS call PGWT. PGWT prevents a process change and preserves the entire program context.

Overview: C/C++ calls of the UTM-XML interface

C/C++ call	Description
KXLInitEnv	Initialize the UTM-XML environment
KXLFrom*datatype*	Convert from *datatype* to the t_value structure (*datatype*= Short, Int, Long, Float, Double, Char, String, Struct, Array)
KXLTo*datatype*	Convert from the t_value structure to *datatype* (*datatype*= Short, Int, Long, Float, Double, Char)
KXLCreateNewObj KXLWrite	Create an XML object
KXLSetSubObject KXLSetRootNode KXLSetParentNode	Navigate in the XML object

C/C++ call	Description
KXLRead KXLReadNode KXLReadNextSib KXLReadChild KXLReadNextSingleNode KXLReadAttr	Read an XML object
KXLConvDocToObj KXLConvObjToDoc KXLFreeObj	Convert between XML object and XML document, release (free) an object
KXLGetHomeEnc KXLGetDocEnc KXLSetDocEnc KXLStringFromUTF8 KXLStringToUTF8	Determine/set character set and convert from/to UTF8
KXLWriteNS KXLDelNS KXLReadNSList KXLSearchNS	Manage namespaces
KXLGetSizeOfNodeList	Query size of a node list
KXLTSENV KXLGetLastParserError	Diagnostic functions
KXLConvDocToObjAndValid KXLParseSchema KXLParseSchemaFile KXLValidDoc KXLValidDocBuf KXLFreeSchema	Support for schema functionality (converting, parsing, validating, freeing)

COBOL interface

The COBOL interface is called via an adapter module. When it is called, the desired function is passed in the form of parameters and a COPY element is used to describe the parameter area. This area must be supplied prior to the call. The parameters correspond to the parameters of the associated C function.

The COBOL interface, in contrast to the C/C++ interface, contains the additional function *KXLSchemaGetRoot* with which the root node of the schema object can be requested. In C/C++ this can be implemented directly using language resources.

You will find more information on the UTM-XML interface in the manual „XML for openUTM“.

8 Generating UTM applications

To use a UTM application, you must define the configuration and generate the application program. This entire procedure is known as “generation”.

In figure 31 in chapter "Creating the application program" you will see an overview of the steps involved in generation. The generation procedure is described in detail in the openUTM manual “Generating Applications”.

It is necessary to take certain specific characteristics into account when generating a UTM cluster application. For further information on UTM cluster applications, see section “UTM cluster application on Linux, Unix or Windows-Systems” and chapter “High availability and load distribution with UTM cluster applications on Linux, Unix or Windows systems”.

For a detailed description of the steps involved during generation, refer to the openUTM manual “Generating Applications”.

8.1 Defining the configuration

To run the application program, you must provide information on the following:

- application properties (maximum values, timers etc.)
- name and properties of user IDs
- system and data access control
- name and properties of clients and partner servers
- name and properties of transaction codes and program units
- reservations for dynamic configuration
- properties of UTM cluster applications

This information is known collectively as configuration information, and is stored in the KDCFILE. The KDCFILE can consist of more than one file (see "The KDCFILE - the “application memory”"), and is created using the **KDCDEF generation tool**.

In addition to the KDCFILE, KDCDEF creates the source text for the ROOT tables. These ROOT tables contain allocation information required internally when using the application.

You have the choice of producing the KDCFILE and the ROOT table sources during a single KDCDEF run, or separately in different KDCDEF runs. If two KDCDEF runs are used, then both runs must receive the same input data.

You must provide KDCDEF with an input file containing KDCDEF control statements that describe the desired configuration.

8.1.1 Overview: KDCDEF control statements

Statement	Function
ABSTRACT-SYNTAX	Define the abstract syntax (OSI TP)
ACCESS-POINT	Create an OSI TP access point for a local UTM application
ACCOUNT	Define accounting parameters
APPLICATION-CONTEXT	Define the application context (OSI TP)
AREA	Define names for additional data areas
BCAMAPPL	Define additional application names
CON	Define a logical connection (LU6.1) to UTM partner applications
CREATE-CONTROL STATEMENTS	Create control statements for a new KDCDEF run from configuration information in the existing KDCFILE
EXIT	Define event exits
HTTP-DESCRIPTOR	Define an HTTP descriptor
KSET	Define a key set
LPAP	Assign an LPAP name as a logical access point for partner applications (LU6.1)
LSES	Define a session name for distributed processing via LU6.1
LTAC	Assign local names for TACs in UTM partner applications
LTERM	Define an LTERM partner as the logical access point for clients and printers
MASTER-LU61-LPAP	Define master LPAP of an LU6.1 LPAP bundle
MASTER-OSI-LPAP	Define the master LPAP of an OSI LPAP bundle
MAX	Define the name of the UTM application and runtime parameters
MESSAGE	Describe the message module
MSG-DEST	Define user-specific message destinations
OSI-CON	Define the logical connection to the partner application (OSI TP)
OSI-LPAP	Define an OSI LPAP name as the logical access point for partner applications (OSI TP)

Statement	Function
PROGRAM	Define the names and properties of program units
PTERM	Define clients and printers
QUEUE	Define table entries for temporary queues
RESERVE	Reserve locations for dynamic configuration
ROOT	Assign names for ROOT table sources
SESCHA	Define session characteristics (LU6.1)
SFUNC	Define special functions for the function keys
SIGNON	Control the sign-on procedure
TAC	Define the names and properties of transaction codes or TAC queues
TACCLASS	Define the number of processes available to TAC classes
TAC-PRIORITIES	Define the priorities for TAC classes
TLS	Define the names of TLS blocks
TPOOL	Define terminal pools
TRANSFER-SYNTAX	Define the transfer syntax
ULS	Define the names of ULS blocks
USER	Insert a user in the configuration
UTMD	Define global application values for distributed processing
The following statements only apply to BS2000 systems:	
CHAR-SET	Assign names for code tables
DATABASE	Describe the database system and the resource manager
DEFAULT	Define default values
EDIT	Define edit options
FORMSYS	Describe the formatting system
LOAD-MODULE	Describe the load module for exchanging programs with BLS
MPOOL	Describe the common memory pool

Statement	Function
MUX	Define a multiplex connection
SATSEL	Define the SAT logging mode and the events to be logged
TCBENTRY	Define a group of TCB entries
The following statements only apply to Unix, Linux and Windows systems:	
CLUSTER	Define global properties of a UTM cluster application
CLUSTER-NODE	Define node application of a cluster
RMXA	Specify a name for a resource manager (database connection via the X/Open XA interface)
SHARED-OBJECT	Define shared objects for a program exchange

There are also statements in addition to those listed above that are used to control theKDCDEF run or insert comments.

8.2 Creating the application program

Before creating the application program, you must write and compile program units. These program units define the application logic.

The application program is created to ensure that the program units can run under the management of openUTM. This involves the following steps:

- compiling the source text for the ROOT tables, which is generated by KDCDEF
- linking the compiled ROOT tables, program units, and any other modules (e.g. format libraries, local message modules, or language-specific runtime systems) to the application program together with UTM modules. This can be done statically (i.e. before the application is started) or dynamically (i.e. when the application is started).

The modules required for your application depends on your application architecture and the operating system platform. Detailed information can be found in the corresponding openUTM manual “Using UTM Applications”.

Main routine KDCROOT

One part of the application program is the main routine KDCROOT. When the application is running, KDCROOT acts as the main control program, and performs the following tasks among others:

- establishes the connection between program units and the UTM system functions
- coordinates the sequence of program units in different programming languages
- interacts with formatting systems (only BS2000 systems)
- sets up links with databases and resource managers

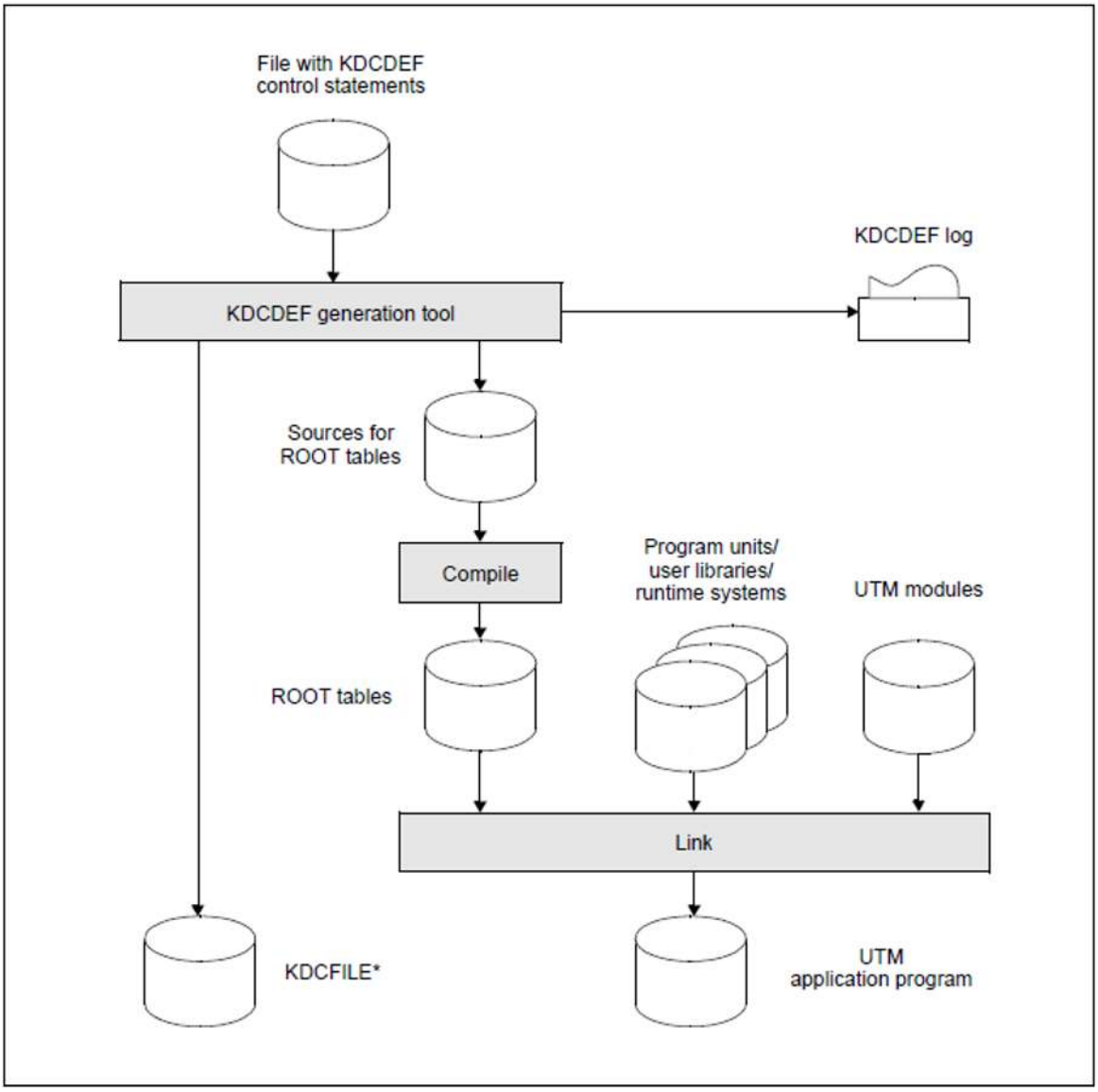


Figure 31: Generating a UTM application

*In a UTM cluster application, further files that are global to the cluster are generated in addition to KDCFILE (see "UTM cluster files").

8.3 Updating the configuration using the KDCUPD tool

With the KDCUPD tool (**KDC**FILE **UPD**ate) you can migrate important user data and administration information after regenerating your UTM application:

- In standalone UTM applications, KDCUPD transfers the data from the old to the new KDCFILE.
- In UTM cluster applications, there is a distinction between node updates (updating the KDCFILE of a single node application) and cluster updates (updating of UTM cluster files).

Furthermore, you can switch with the help of KDCUPD from an older openUTM version to the current openUTM version without the data in the KDCFILE of the previous production application.

KDCUPD in standalone applications

In a standalone UTM application, KDCUPD can be used for both UTM-S and UTM-F.

In the case of UTM-S, KDCUPD enables the user data to be transferred from the previous KDCFILE to the new KDCFILE on completion of the application run. KDCUPD can transfer all user data or only specific user data, as required.

Once all the user data has been transferred by KDCUPD and the application has been restarted, the users can resume their work:

- Interrupted services are continued.
- Processing of UTM-controlled message queues continues:
  - all asynchronous messages are output
  - all background jobs are executed
  - all time-driven jobs are processed by openUTM at the specified time
- All messages in service-controlled queues are available for further processing.

KDCUPD in UTM cluster applications on Linux, Unix and Windows systems

In UTM cluster applications, KDCUPD can be used with both UTM-S and UTM-F. KDCUPD enables you to perform both node updates and cluster updates.

Node updates

In the case of a node update, KDCUPD transfers administration and user data from a node application’s old KDCFILE to a node application's new KDCFILE. Data that applies locally at node level such as TLS, asynchronous messages or background jobs is transferred.

If there are a large number of changes to the generation (for example, new connections to partner applications), it is possible to perform a node update while the UTM cluster application is running (online update). In this case, the node update is performed for each individual node in sequence, i.e. the node application is terminated, the data is transferred using KDCUPD and the node application is then started again. The remaining node applications continue to run during this process.

For some changes, it is necessary to terminate the UTM cluster application.

Cluster updates

In the case of cluster updates, KDCUPD transfers administration and user data from the old UTM cluster files to the new UTM cluster files. To do this, it is necessary to terminate the UTM cluster application.

It is only necessary to perform a cluster update if fundamental changes are made to the cluster configuration, e.g. if the number of generated node applications or the number of cluster page pool files is increased.

During a cluster update, data that is valid globally in the cluster (ULS, GSSB, service data, passwords, locales) is transferred

After a cluster update, it is generally necessary to perform a node update for each node application.

For further information on using KDCUPD in a UTM cluster application, see "Update generation in a cluster" in the platform specific openUTM manual “Using UTM Applications”.

Checking the KDCFILE for consistency

KDCUPD also provides the CHECK option, with which you can check the files of a KDCFILE for consistency without transferring user data to a new KDCFILE.

Migrating with KDCUPD

The UTM tool KDCUPD can also be used to migrate to another version of openUTM. You can transfer user data from the KDCFILE of the preceding versions into the newly created KDCFILE of the current openUTM version. You start the application under the current openUTM version using this KDCFILE, and afterwards the users can continue with their current work. For instance, after upgrading to a follow-up version, a time-driven print job initiated with the preceding version can be automatically activated by the current version - exactly at the desired time.

Migrating to a 64-bit environment with KDCUPD on Linux, Unix and Windows systems

KDCUPD can be used to transfer information from a 32-bit application environment to a 64-bit application environment. This means that you are able to switch over to a 64-bit environment quickly and without loss of data.

You will find more information on KDCUPD in the openUTM Manual “Generating Applications”.

9 Administering UTM applications

openUTM offers comprehensive, easy-to-use administration concepts, which allow for effective, flexible utilization of the UTM applications and provide maximum availability.

The term “administration” covers all activities required to control and monitor active UTM applications, such as:

- defining and modifying the number of parallel processes, timers, scheduling, etc. (tuning the application)
- locking and releasing transaction codes
- integrating new services
- replacing parts of the application program or the entire application program during operation
- adding or removing clients, printers, or user IDs in the configuration (dynamic configuration)
- defining the number of clients permitted or locked for an LTERM pool
- establishing or shutting down logical connections to clients, printers or remote server applications
- switching logging files
- setting diagnostic tools
- displaying the operational data of the UTM application
- determining the IP address and making it available to the UTM application
- generating, reading or deleting RSA keys
- modifying cache properties
- terminating the UTM application

In a UTM cluster application, each of these activities applies only locally to the individual node application or globally to all node applications.

KDCADMI program interface

For administration purposes, openUTM provides the KDCADMI program interface. This interface can be used to create your own administration programs tailored to the respective application. All administration functions can be accessed via the KDCADMI program interface (see "Administration program interface").

KDCADM command interface

Alternatively, openUTM also provides the standard KDCADM administration program, which already contains predefined basic administration functions (see "Administration command interface").

WinAdmin graphical administration program

The openUTM component WinAdmin allows you to administer one or more UTM applications on a PC with the comfort of a graphical user interface - even if the UTM applications in question are running on different platforms distributed across the network (see also "WinAdmin graphical administration program").

WinAdmin covers the entire range of functionality available through KDCADMI and also offers additional functions as well. WebAdmin is also available as an add-on on a management unit of an SE Server.

WebAdmin graphical administration program

Like WinAdmin, WebAdmin is a program which is used to administer UTM applications and possess a graphical user interface. Unlike WinAdmin, however, WebAdmin permits Internet-based administration. A single WebAdmin instance is installed centrally and provides a web application that users can sign on at from any web browser, see also "WebAdmin graphical administration program".

WebAdmin covers the entire function scope available with KDCADMI.

Administration with CALLUTM on BS2000 systems

CALLUTM is an UTM client program on BS2000 systems that can be used in many applications and with which you can administer one or more local or remote UTM applications via an SDF interface. These applications can be located on different platforms. You will find more information on the possible areas of application for CALLUTM on "BS2000-specific functions".

The KDCADMI administration program interface, the standard KDCADM Administration program and the CALLUTM program are described in detail in the openUTM manual “Administering Applications”. The following sections merely provide a brief overview. Further information on openUTM WinAdmin or openUTM WebAdmin can be found in the respective extensive online help systems and in the User Guides. Both are delivered with WinAdmin and WebAdmin.

9.1 Administration command interface

The basic administration functions are called by means of predefined transaction codes, which are allocated to the standard administration program (KDCADM) during generation. These predefined transaction codes are known as **administration commands**.

Each basic function has a dialog transaction code and an asynchronous transaction code. The basic functions can thus be executed in a dialog, or asynchronously via message queueing.

Administration in a dialog

Within the dialog, the basic administration functions are initiated synchronously either by the administrator at the terminal or by a client program. In both cases, the desired administration command is specified in the form of a dialog transaction code. openUTM immediately executes the requested administration function, and returns a corresponding response.

Several administrators or administration clients can access the administration functions simultaneously.

Administration via message queueing

Like administration in a dialog, this type of administration allows the basic administration functions to be initiated by the administrator at the terminal. In this case, however, the desired command is entered in the form of an asynchronous transaction code.

It is also possible to access the basic administration functions from UTM program units. For this purpose, the program unit issues an MQ call (FPUT or DPUT) for the corresponding asynchronous transaction code.

In both cases, openUTM places the administration job in the appropriate queue, executes it independently of the administrator or program unit, and outputs the result in the form of an asynchronous message to a defined destination. Possible destinations include the administrator’s terminal, another terminal, a printer, or an asynchronous program.

With administration via message queuing, several administrators or UTM program units can access the administration functions simultaneously.

If administration is initiated by means of MQ calls from UTM program units, the time control facility can be used.

Overview: Transaction codes of the standard administration program

Dialog TAC	Asynchronous TAC	Administration function
KDCAPPL	KDCAPPLA	Change the number of processes, timer settings, and maximum values; replace the application program; switch logging files, enable/disable accounting functions in openUTM on BS2000 systems
KDCBNDL	KDCBNDLA	Exchange the master LTERMs of two LTERM bundles.
KDCDIAG	KDCDIAGA	Call diagnostic tools: enable/disable test mode, trace, and KDCMON functions, request dumps Enable and disable the debug mode for the XA database interface. In openUTM on BS2000 systems, STXIT logging can be enabled and disabled as well.
KDCHELP	KDCHELPA	Request information on the syntax of TACs from KDCADM
KDCINF	KDCINFA	Query the current settings of system parameters, load statistics for the application, and object properties
KDCLOG	KDCLOGA	Switch the user log file to the next file generation
KDCLPAP	KDCLPAPA	For administration of UTM applications for distributed processing: establish / shut down logical connections to partner applications, switch alternate connections to OSI TP partners, lock/unlock partners, change the timer for monitoring sessions/associations
KDCLSES	KDCLSESA	For administration of UTM applications for distributed processing: establish / shut down logical connections for a session
KDCLTAC	KDCLTACA	For administration of UTM applications for distributed processing: lock/unlock a remote service (LTAC) for the local application, set the timer for monitoring the session/association setup and the response times of the partner service
KDCLTERM	KDCLTRMA	Lock/unlock an LTERM partner, establish / shut down connections, assign an LTERM to an LTERM group.
KDCPOOL	KDCPOOLA	Change the number of clients permitted for a terminal pool
KDCPROG	KDCPROGA	Replace the load modules of the application program
KDCPTERM	KDCPTRMA	Lock/unlock clients/printers, establish / shut down connections
KDCSHUT	KDCSHUTA	Terminate the application In the case of a UTM cluster application: May apply to a node application or the entire UTM cluster application as required.

Dialog TAC	Asynchronous TAC	Administration function
KDCSLOG	KDCSLOGA	Switch the system log file (SYSLOG) of the application, enable/disable size monitoring, define a threshold value for size monitoring, query information on the SYSLOG
KDCSWTCH	KDCSWCHA	Change the allocations between the client/printer and the LTERM Partner
KDCTAC	KDCTACA	Lock/unlock transaction codes (local services)
KDCTCL	KDCTCLA	Change the number of processes available to a TAC class
KDCUSER	KDCUSERA	Lock/unlock user IDs, change passwords
The following TACs are only available in openUTM on BS2000 systems:		
KDCMUX	KDCMUXA	Lock/unlock multiplex connections, establish / shut down connections
KDCSEND	KDCSENDA	Send messages to terminal users

9.2 Administration program interface

You can use the KDCADMI program interface provided by openUTM to create your own administration programs specially tailored to your application. Since the administration program interface provides powerful functions which can be used individually to program your own administration programs, user-defined administration programs offer more options than the basic administration functions:

- Practically all generation information is available.
- On this basis, you can query, analyze and further process the information that is of practical interest to you.
- Dynamic configuration calls can be used in user-defined administration programs (see below).
- Formats can be used for administration dialogs.

The program interface calls are independent of the platform on which the administration program is running. It is thus possible, for example, to administer one or more UTM applications running on Unix or Linux systems or BS2000 systems from a UTM application on a Windows system, and vice versa. Since compatibility with future openUTM versions is also guaranteed, user-defined administration programs need not be adapted if you decide to switch to another platform or upgrade to new openUTM versions.

The effort involved in creating your own administration programs is minimal:
the program interface calls can be integrated into C ,C++ or COBOL program units. Both dialog and asynchronous programs are supported, and a program can contain an unlimited number of administration calls. The data structures required are already predefined and are provided in the form of header files/COPY elements.

Dynamic configuration

The KDCADMI program interface offers calls for modifying the application configuration “on-the-fly”. Clients, printers, user IDs, services, et cetera, can be added to or removed from the configuration during operation, without affecting system availability.

Corresponding KDCDEF statements can be created - online or offline - for all dynamically configurable objects (inverse KDCDEF). These statements are then provided as input for the KDCDEF generation tool, which means that all dynamic changes to the configuration can be incorporated without any problems during regeneration.

Overview: Administration functions of the KDCADMI program interface

Operation code	Function
KC_CHANGE_APPLICATION	Replace the entire application program during operation
KC_CREATE_DUMP	Create a UTM dump
KC_CREATE_OBJECT	Add new objects (program units, terminals, users, etc.) to the configuration dynamically
KC_CREATE_STATEMENTS	Create a KDCDEF control statement for dynamically configurable objects during operation (online)
KC_DELETE_OBJECT	Delete application objects, i.e. remove them from the application configuration
KC_ENCRYPT	Create, delete or read an RSA key pair
KC_GET_OBJECT	Request information on the objects and parameters of the application
KC_LOCK_MGMT	Remove cluster user file lock
KC_MODIFY_OBJECT	Modify the properties of objects or application parameters
KC_ONLINE_IMPORT	Import application data online
KC_PTC_TA	Roll back transaction in state PTC
KC_SHUTDOWN	Terminate the application
KC_SPOOLOUT	Automatically establish a connection to printers for which messages are available
KC_SYSLOG	Administer the SYSLOG system log file
KC_UPDATE_IPADDR	Update the IP address
KC_USLOG	Switch the user log file(s) to the next file generation during operation
The following operation code is only available on BS2000 systems:	
KC_SEND_MESSAGE	Send a system line message to one or more dialog terminals

Sample programs

openUTM supplies the following C sample programs which demonstrate how to use the KDCADMI interface:

- SUSRMAX (show Users and MAX parameter)
- ENCRADM (encryption administration)
- ADJTCLT (adjust tacclass tasks)
- HNDLUSR (handle user data, on BS2000 systems only)

For COBOL openUTM supplies the sample program COBUSER. Since these sample programs are provided also in the form of source code, they can be individually adapted or used as templates for your own administration programs. Alternatively, they can be used without modification, e.g. to query information on user IDs and maximum values, to change current settings, or to dynamically configure user IDs.

For Unix, Linux and Windows systems the sample programs ENCRADM, SUSRMAX and ADJTCLT are integrated in the sample application or the Quick Start Kit, the sample program COBUSER is supplied in the *utmpath* under *sample/src* or *sample\src*. For BS2000 systems all sample programs are included in the library SYSLIB.UTM.070.EXAMPLE.

You will find further information regarding the sample programs at the beginning of the respective source code.

9.3 WinAdmin graphical administration program

WinAdmin is a graphical administration terminal from which you can administer several UTM applications at the same time.

WinAdmin is based on Java technology and can therefore run on Windows systems as well as on Unix platforms and Linux platforms.

The UTM applications can run on all platforms released, and the programs can be of different versions. The administration functions available for the individual UTM applications depends on the openUTM version:

- For UTM applications the full range of functions of the KDCADMI program interface offered by the relevant openUTM version can be used (see section “Administrationprogram interface”). You can, for example, add objects dynamically to a configuration or delete objects see section "Changing the generation dynamically". The functions are listed on "Administration program interface".

  In addition, you can start UTM applications with WinAdmin. This requires that openFT is used on the participating computers.
- Furthermore, WinAdmin offers the following functions that are not accessible via KDCADMI:
  - definition of message collectors. This makes it possible for WinAdmin to query, display and archive UTM messages of UTM applications that are currently running.
  - administration of message queues
  - printer administration and control
  - display of GSSB contents and deletion of GSSBs
  - creation and deletion of temporary queues
  - grouping of several administration steps into a single transaction
  - far-reaching support for the UTM security concept by means of roles and access lists (see "Access list concept")
  - definition of actions, e.g. time-controlled storage of object properties in files or responding to threshold violations.
  - collection and archiving of statistical data on the UTM applications.

Graphical interface

The ease of use of a Windows interface introduces significant administration advantages, especially for complex applications. Examples of such advantages are:

- Simple navigation:
  You can quickly find the desired object or a certain application with a click of the mouse.
- Clarity:
  The parameters of an object, such as a client or a user, are displayed in an easy-to-read list in a window, and the parameters can be changed there.
- Tables
  Objects of the same type, such as printers or LTERMs, are displayed in easy-to-read tables and can be sorted with a click of the mouse.
- Diagrams:
  Statistics can be presented graphically, e.g. the number of transactions per second within a specific interval. The statistical values can be easily saved to a file and used later in an analysis.

- Plausibility checks:
  Relationships between different objects, e.g. between user IDs and key sets, are considered in their respective context.

Administering several applications

WinAdmin allows a single point of view for several applications by combining these applications in collections. You can, for example, create a collection of all branch applications and administer these applications together. These applications are viewed separately in the central office.

Administering UTM cluster applications on Linux, Unix and Windows systems

WinAdmin provides administration functions which you can apply globally to all of the node applications in the UTM cluster application. Furthermore, WinAdmin allows you, for example, to display statistical summaries which include all the running node applications.

For more detailed information on administering a UTM cluster application, see section “UTM cluster application on Linux, Unix and Windows systems” and „WinAdmin Online-Hilfe“.

Application view

In this classical view you see a certain application on a certain computer with its settings and objects. You can then work from the “top down” and view or change specific parameters, or you can delete or recreate objects such as users and clients.

Object view

In this view, you select objects from several different applications in any order you want, e.g. all users in a collection. You can then separate them according to their server or application and output all locked users for several applications. You can then unlock these users in a single action.

9.4 WebAdmin graphical administration program

The WebAdmin component is a web-based application for the administration of UTM applications on all platforms.

The Apache Tomcat web server, in which the web application is automatically deployed on initial start-up, is supplied together with WebAdmin. WebAdmin can be installed either as a stand-alone tool or as an add-on on a management unit of an SE Server. Broadly speaking, the two variants provide the same function scope.

Once you have performed central installation of WebAdmin, it provides a web application which you can access from any client computer. A web browser simply needs to be present on the client computer.

The UTM applications can be running on any released platform and be of different versions. The administration functions that are possible for the individual UTM applications depends on the openUTM version.

- When administering UTM applications, you are able to use the full function scope of the KDCADMI program interface provided by the openUTM version in question, see section “Administration program interface”. Thus, for example, you can dynamically add objects to a configuration or delete objects, see also "Changing the generation dynamically". The functions are listed on "Administration program interface".

  If you are using WebAdmin as a stand-alone tool then you can also use it to start UTM applications. For this to possible, openFT must be used at the participating computers.
- WebAdmin additionally offers the following functions that are not available via KDCADMI:
  - Definition of message collectors. This enables WebAdmin to poll UTM messages from the running UTM applications and to display and archive these.
  - Administration of message queues
  - Printer administration and printer control
  - Creation and deletion of temporary queues
  - Display of GSSB contents and deletion of GSSBs
  - Very broad-based support for the UTM security concept by means of roles and access lists, see "Access list concept"
  - Definition of actions, e.g. time-controlled storage of object properties in files or responses when values rise above or fall below certain thresholds.
  - Collection and archiving of statistical data for UTM applications.

  Unlike WinAdmin, WebAdmin offers "round the clock" monitoring of UTM applications by means of statistics collectors and, in some cases, threshold actions. For this to be possible, it is not necessary for a client to be connected to the Web application. WebAdmin periodically checks the availability of the monitored UTM applications.

Graphical user interface

In particular when dealing with complex applications, the convenience of a web-based graphical user interface offers considerable advantages during administration tasks, such as:

- Ease of navigation:
  Required objects or specific application parameters can be located rapidly at the click of a mouse button.
- Clear overview:
  The parameters relating to an object such as a client or user are clearly listed on a tab where they can also be modified.

- Tables:
  Objects of the same type such as printers or LTERMs are listed in clearly presented tables where they can be sorted at the click of a mouse button.
- Diagrams:
  Statistics can be presented graphically, e.g. the number of transactions per second within a specific interval. The statistical values can be easily saved to a file and used later in an analysis.
- Plausibility checks:
  Relationships between different objects, e.g. between user IDs and key sets, are considered in their respective context.

Administering UTM cluster applications on Linux, Unix and Windows systems

WebAdmin provides administration functions which you can use with all of the node applications in the UTM cluster application. Furthermore, WebAdmin allows you, for example, to display statistical summaries which include all the running node applications.

For detailed information on administering a UTM cluster application, see section “UTM cluster application on Linux, Unix and Windows systems” and the „WebAdmin Online-Hilfe“.

9.5 Authorization concept

In addition to the general security functions, openUTM provides a two-level authorization concept specifically for administration.

- Level 1: Read access to all administrative data

  If ADMIN=READ is assigned to a transaction code of an administration program during configuration, the corresponding program is granted read access to all administrative data. The user does not require administration authorization to call this transaction code. The information is thus accessible to all users.

- Level 2: Full administration authorization

  To gain unrestricted access to all administration functions (commands and user-defined administration programs), the following requirements must be met:

  - ADMIN=Y must be set during configuration for a transaction code that calls an administration program.
  - The user calling this transaction code must have administration authorization, i.e. PERMIT=ADMIN must be set for the user ID and the partner application during configuration.

It is also possible to define more subtle differentiations using the data access control mechanism of the lock/key code concept (see "Data access control (authorization)").

9.6 Changing the generation dynamically

You can change the static UTM generation (created using KDCDEF) dynamically without interrupting the application run. The WinAdmin and WebAdmin graphical administration programs are available to you for this purpose. These allow you to modify objects in the generation, add new objects to it or delete objects from it. However, you can also write your own administration programs on the basis of the KDCADMI program interface.

The following table provides an overview of which object types of a UTM generation can be created, modified and deleted in the current version of UTM (Cluster-related object types are only available on Linux, Unix and Windows systems):

KDCDEF statement and operand for object type	Meaning of the object type	Create	Modify	Delete
CLUSTER	Parameters for a UTM cluster application	No	Yes	No
CLUSTER-NODE	Address components and base names of a node application	No	Yes	No
CON	LU6.1 connections	Yes	No	Yes
DATABASE/DB_INFO (BS2000 systems)	Database systems	no	yes	no
KSET (+ LOCK operand)	Access authorizations (key sets) (+ access protection)	Yes	Yes	Yes
MAX	Application parameters and maximum values	--*	Yes*	--*
LOAD-MODULE (BS2000 systems)	Load modules	No	Yes	No
LPAP	LPAP partners for LU6.1	No	Yes	No
LSES	Sessions for LU6.1	Yes	Yes	Yes
LTAC	TACs for LU6.1 partners and OSI TP partners	Yes	Yes	Yes
LTERM	LTERM partners of clients and printers	Yes	Yes	Yes
MUX (BS2000 systems)	Multiplex connection	No	Yes	No
OSI-CON	Connection for OSI TP partners	No	Yes	No
OSI-LPAP	OSI LPAP partners for OSI TP	No	Yes	No
PROGRAM	Program units	Yes	No	Yes
PTERM	Clients and printers	Yes	Yes	Yes
SHARED-OBJECT (Unix, Linux and Windows systems)	Shared objects	No	Yes	No
TAC	Transaction code and TAC queues	Yes	Yes	Yes

TACCLASS	TAC classes	No	Yes	No
TPOOL	LTERM pools	No	Yes	No
USER	User IDs	Yes	Yes	Yes

* It does not make sense to create or delete the application parameters. The openUTM manual “Administering Applications” and the WinAdmin and WebAdmin online help systems describe which parameters can be changed.

It is also possible to delete RSA keys and create new keys. By default, RSA keys are created in the KDCDEF run in accordance with the entries for PTERM, TAC and TPOOL.

All dynamic changes to the UTM generation are subject to transaction management. In other words, the change is carried out either in its entirety or not at all. If you use WinAdmin, you can also group a number of steps together in a transaction (change the properties of multiple users at once, for example).

When objects are to be created dynamically, sufficient storage space must be reserved at generation with KDCDEF.

The openUTM manual “Administering Applications” tells you which objects and parameters can be changed dynamically and how this is done (look for information on the KDCADMI program interface and transaction management).

9.7 Automated administration

If you wish to use the event management options provided by openUTM, you can automate administration activities as described below.

There are two ways of doing this.

- openUTM signals a certain event (e.g. the loss of a connection) by sending a UTM message to the destination MSGTAC. This message automatically activates the MSGTAC event service. The MSGTAC routine reads in the message, evaluates it, and initiates the appropriate administration function.

  The following preparations are required here:

  - Assign the destination MSGTAC to all UTM messages relevant for administration. The UTM tool KDCMMOD (**KDC M**essage **MOD**ify) is provided for this purpose.
  - Define the MSGTAC routine as a KDCS program unit into which messages are read using the KDCS call FGET. Evaluation of the messages is easy to program, since openUTM provides data structures that correspond to the structure of the messages (in the COPY element KCMSGC for COBOL, and in the header file *kcmsg.h* for C/C++). Use FPUT or DPUT calls to initiate the corresponding administration functions.
  - During generation, add the modified message module to the configuration using the KDCDEF statement MESSAGE, and assign the transaction code KDCMSGTC to the MSGTAC routine using the KDCDEF statement TAC.

    Link the compiled message module and MSGTAC routine into the application program.

    The openUTM manual „Programming Applications with KDCS” contains further information on the MSGTAC event service, as well as sample MSGTAC routines in C and COBOL.

- Automatic administration is possible using the message destinations USER-DEST-n (n=1,2,3,4) in exactly the same way as using MSGTAC.
  - Assign an asynchronous TAC to a message destination USER-DEST-n using the KDCDEF statement MSG-DEST and then assign the message destination USER_DEST-n to the relevant UTM messages using the KDCMMOD utility program.
  - You can then read the messages and issue administration calls in the program assigned to the asynchronous TAC – in the same way as in the MSGTAC routine.

9.8 Administering message queues and printers

openUTM provides the KDCS call DADM (**D**elayed free message **ADM**inistration) for administering UTM-controlled and service-controlled message queues. This call allows you to:

- request an overview of the contents of a queue
- request specific information on individual jobs in the queue
- bring individual jobs forward
- delete individual jobs
- delete all the jobs in a queue
- move messages from the dead letter queue.

The KDCS call PADM (**P**rinter **ADM**inistration) is provided for administering printers, and offers the following options:

- confirm or repeat print jobs
- switch between acknowledgment mode and automatic mode
- request information on a printer or print job
- change a printer assignment
- lock and unlock printers, establish and shut down connections to printers

These calls are implemented via the KDCS program interface, and not via the administration interface. Full administration authorization is therefore not essential for day-to-day activities such as the cancellation of print jobs or the confirmation of important printouts such as checks: during generation, you can define print control LTERMs to allow users to administer the printers and print queues they normally use without requiring administration authorization. Administration authorization will, however, still be required for the administration of “exotic” printers and queues.

openUTM supplies the sample programs KDCDADM and KDCPADM, which provide direct access to all features of the DADM and PADM calls.

Message queues and printers can also be administered using WinAdmin ("WinAdmin graphical administration program") or WebAdmin ("WebAdmin graphical administration program").

Further information on administering message queues and printers, and on the sample programs, can be found in the openUTM manual “Administering Applications”.

10 Security functions

Enterprise-wide, integrated IT solutions would be inconceivable without appropriate security functions. openUTM offers comprehensive, distinct, clearly structured security concepts, which allow for open solutions in situations where these were previously not possible for security reasons.

The openUTM security functions include:

- system access control functions (identification, authentication), even with client/server communication and distributed processing via OSI TP
- data access control functions (authorization)
- system and data access control, even with distributed processing
- encryption of passwords and user data for client/server communication
- an additional two-level authorization concept specially designed for administration (see section “Authorization concept”)
- the option of using the security mechanisms of external resource managers

10.1 System access control (identification and authentication)

openUTM offers the following system access control options:

- definition of logical access points for clients and partner servers
- user IDs and passwords when using terminals
- user IDs and passwords when using client programs
- user IDs and passwords when using OSI TP partner applications (cross-application user concept)
- system access control by means of ID cards when using terminals
- use of Kerberos on BS2000 systems when using terminals
- silent alarm in the case of repeated failed attempts to gain access
- user-defined sign-on checks - SIGNON event service

These options are explained briefly in the following sections. The precise format of the corresponding generation statements can be found in the openUTM manual “Generating Applications”. The KDCADMI calls are described in detail in the openUTM manual “Administering Applications”.

Definition of logical access points for clients and partner servers

All clients and partner servers who wish to use a UTM application must be known to the application. This is achieved by assigning a logical access point to the client or partner server, which has been defined in the UTM application configuration.

LTERM partners - access points for clients

The logical access points for clients are known as LTERM partners (**L**ogical **TERM**inal), and are generated using the KDCDEF statement LTERM. The KDCDEF statement PTERM (**P**hysical **TERM**inal) is used to assign a “real” client to the LTERM access point. LTERM partners and PTERM assignments can also be defined dynamically while the application is running (KCADMI call KC_CREATE_OBJECT).

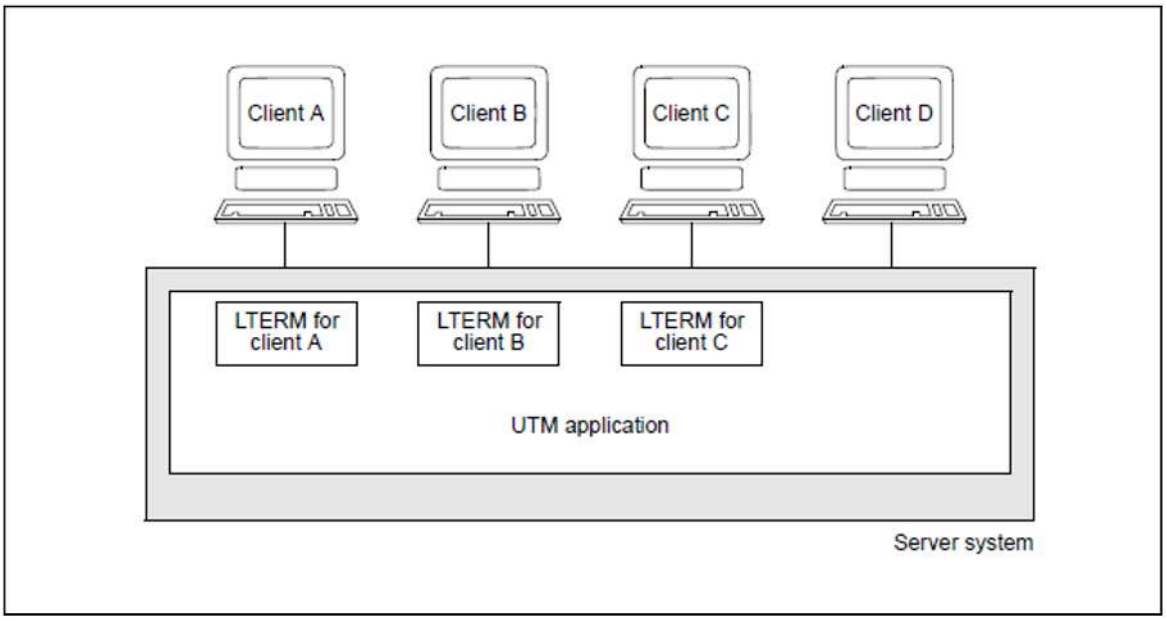


Figure 32: Connecting clients via LTERM partners

In the sample configuration in figure 32, clients A, B, and C can work with the UTM application. Although client D has a line to the server system, it cannot access the UTM applications, since it has not been assigned an LTERM partner in the application configuration.

If you wish, you can make this strict assignment system more flexible through the use of LTERM pools. LTERM pools allow you to provide a defined number of clients with access to the UTM application without explicitly assigning an LTERM partner to each client. If you use an LTERM pool, each client that wishes to connect to the UTM application and that has not been generated explicitly is automatically assigned an LTERM partner from the pool.

(OSI) LPAP partners - access points for partner servers

The logical access points for partner servers are known as LPAP partners (**L**ogical **P**artner **AP**plication) or, when using the OSI TP protocol, OSI LPAP partners. They are defined accordingly using the KDCDEF statements LPAP or OSI-LPAP. The KDCDEF statements CON (**CON**nection) and OSI-CON can be used to assign the “real” partner application.

Definition of user IDs and passwords

It is possible to define user IDs for UTM applications either during generation using the KDCDEF statement USER, or dynamically using the KDCADMI call KC_CREATE_OBJECT.

If a UTM application is generated with UTM user IDs, then it is possible to define user-specific passwords.

When defining passwords, it is possible to impose conditions, such as a minimum length and a certain level of complexity. You can also set the minimum and maximum validity period for the password of a user.

The passwords and user IDs are stored in an encrypted form by openUTM, i.e. they cannot be recognized in a dump. Passwords are transmitted in an encrypted form when communicating with clients and terminal emulations if they support encryption.

Users can change their own password while the system is running if a corresponding service has been created for the UTM application.

User IDs and passwords when using terminals

All terminal users who wish to work with a UTM application must identify themselves to the application by specifying their user ID. This sign-on check is also known as KDCSIGN. If a password has also been generated, then the password must also be specified.

Terminal users can change their password themselves when signing on (only possible for hidden passwords in UTM (BS2000)).

User IDs and passwords when using client programs

The openUTM identification and authentication concept is also available when using terminals and UTM client programs.

UPIC clients and OpenCPIC clients

UPIC clients and OpenCPIC clients transfer user IDs and passwords to the UTM application by means of special calls:

- With CPI-C, the calls *Set_Conversation_Security_User_ID* and *Set_Conversation_Security_Password* are used.
- With XATMI, the *usrname* and *passwd* parameters of the *tpinit* call are used.

When using UTM client with the UPIC carrier system, you can also change the password *(Set_Conversation_Security_New_Password call).* The client can determine whether the validity period of a password has expired by means of a return code.

Before starting a service, openUTM validates the authorization data transferred by the client, and assigns the respective user ID and the corresponding authorization profile (see "Data access control (authorization)"). This is similar to the KDCSIGN of a terminal user.

A single client program can thus run under several different user IDs with their own individual authorization profiles.

TS applications

The identification and authentication concept of openUTM is only available to you when using transport system applications if you have defined a sign-on service for this transport system access point (see "System access control (identification and authentication)").

The sign-on is valid for TS applications as long as the connection is available. When the connection is cleared, openUTM starts the sign-on service, validates the authorization data passed by the sign-on service, and assigns the corresponding user ID and the corresponding authorization profile for the duration of the connection.

If a client program does **not** transfer any authorization data, then a connection user ID permanently assigned to the LTERM partner is signed on. Clients with no protocols or interfaces for transferring authorization data (e.g. transport system applications) can thus access UTM applications - but only with a authorization profile of the connection user ID.

Detailed information on the security concept when connecting client programs can be found in the openUTM-Client manuals.

User concept for server-server communication via OSI TP

openUTM's user concept is available globally in the context of server-server communication via OSI TP. When addressing the partner, you can select a security type in the APRO call:

N (none)
No authorization data will be passed to the job receiver.

S (same)
The user ID under which the local service is running will be passed to the job receiver.

P (program)
Values specified explicitly in the program as the user ID and password will be passed to the job receiver.

System access control by means of ID cards

The user IDs for a UTM application can be configured, such that a special ID card is required to access the application. For this purpose, a corresponding reading device must be available at the terminal. If the ID card is removed from the reader while the UTM application is running, openUTM shuts down the connection to the terminal. In the case of terminals with appropriate reading devices, access to the UTM application can thus be controlled by means of a user ID, a password, and a valid ID card.

Using Kerberos with openUTM on BS2000 systems

For terminals, openUTM on BS2000 systems together with SECOS permits the use of Kerberos (RFC1510). Kerberos is network authentication protocol developed at the Massachusetts Institute of Technology (MIT). It is a security system based on a cryptographic encryption schemes. When using Kerberos for authentication, passwords are not passed as plain text over the network. This prevents passwords from being intercepted in the network. Furthermore, there is no need to administer the passwords for a user for different applications, i.e. Kerberos permits a single sign-on for different applications.

Kerberos works with symmetric encryption, which means all keys are available at two locations; one with the owner of a key (principal) one at the KDC (key distribution center).

Silent alarm in the case of repeated failed attempts to gain access

Following several consecutive failed attempts on the part of a client or OSI TP partner to sign on to a UTM application, openUTM sends a message internally to the default destination SYSLOG (system log file) or to MSGTAC (optional) to signal the possibility of illegal access attempts. The sign-on attempts of a user do not need to be performed on the same client. Unsuccessful user sign-on attempts can therefore also be monitored when access is performed via a terminal pool.

You are thus given the opportunity to take appropriate measures, e.g. sign off the connection by means of automatic administration (see "Automatic administration"). In the configuration, you can the define the number of permitted failed access attempts, after which this message is to be generated.

Automatic close of connection

During generation, you can define the maximum time that the UTM application should wait for terminal input after the end of a transaction - or after dialog output during a transaction. If no data is entered within this period, then the connection to the client is cleared down. If the client is a terminal, then a message is also output. If terminal users forget to sign off from the UTM application after completing their work, for instance, the automatic timeout mechanism reduces the possibility of unauthorized access to the UTM application.

User-defined sign-on checks - SIGNON event service

When signing on to a UTM application, predefined UTM messages are output by default requesting the terminal user to enter a user ID and password, and if necessary, to insert an ID card.

However, you can use the SIGNON event service to customize the sign-on dialog for your application, and to introduce your own authorization checks to supplement those of openUTM. You can define multiple SIGNON services and thus design sign-on services specifically for particular types (terminals, clients with the UPIC carrier system, transport system applications, etc.).

openUTM supplies sample programs for a SIGNON service - both the compiled objects and the COBOL sources. This SIGNON service, which implements a sign-on dialog with formats, can be used without modification or adapted for your own dialogs.

Information on how to program and operate SIGNON services can be found in the openUTM manual „Programming Applications with KDCS”.

10.2 Data access control (authorization)

UTM applications generally comprise a large number of services. Some of these must be available to all users, while others must only be accessible to certain users. In the case of services that have access to sensitive data, it makes sense to restrict access to a few selected users. In addition, access can be further restricted by permitting security-relevant accesses only via specific LTERM partners (access points). openUTM therefore offers you the opportunity of specifying subtly differentiated, multi-level access rights in the configuration of a UTM application.

openUTM offers you two access control methods for this. These offer the same options for differentiation but use different concepts for viewing the UTM objects:

- the user-oriented lock/key code concept
- the role-oriented access list concept

The two methods can be combined in a single application, but you must opt for one of the two methods for a specific object.

10.2.1 Lock/key code concept

The lock/key code concept allows you to determine, for example, that only users or client programs with a particular type of authorization can use particular services of the UTM application UTM. You can also specify that signing on under a user ID is only possible via certain LTERM partners (access points) or that certain services can only be started via specific LTERM partners. It is thus possible to restrict the authorization of a user to start a particular service to the use of a specific, particularly secure terminal or client system.

The objects to be protected (e.g. LTERM partners, transaction codes assigned to services) are provided with a lock code, which takes the form of a logical number lock. Key codes are defined for user IDs and LTERM partners. If a key code matches the lock code of a protected object, access is granted to this object.
A user ID or an LTERM partner generally has access to a number of services, and therefore has several key codes. The individual key codes are combined to form key sets.

The lock/key code concept results in the following effects:

- A terminal user or a client program can sign on only if the specified user ID is assigned a key code that matches the lock code of the relevant LTERM partner.
- A terminal user or a client program can call a particular service only if the key sets of **both** the respective user ID **and** the LTERM partner contain a key code that matches the lock code of the transaction code.

The diagram on the next page shows an example of how to use the lock/key code concept.

Example of how to use the lock/key code concept

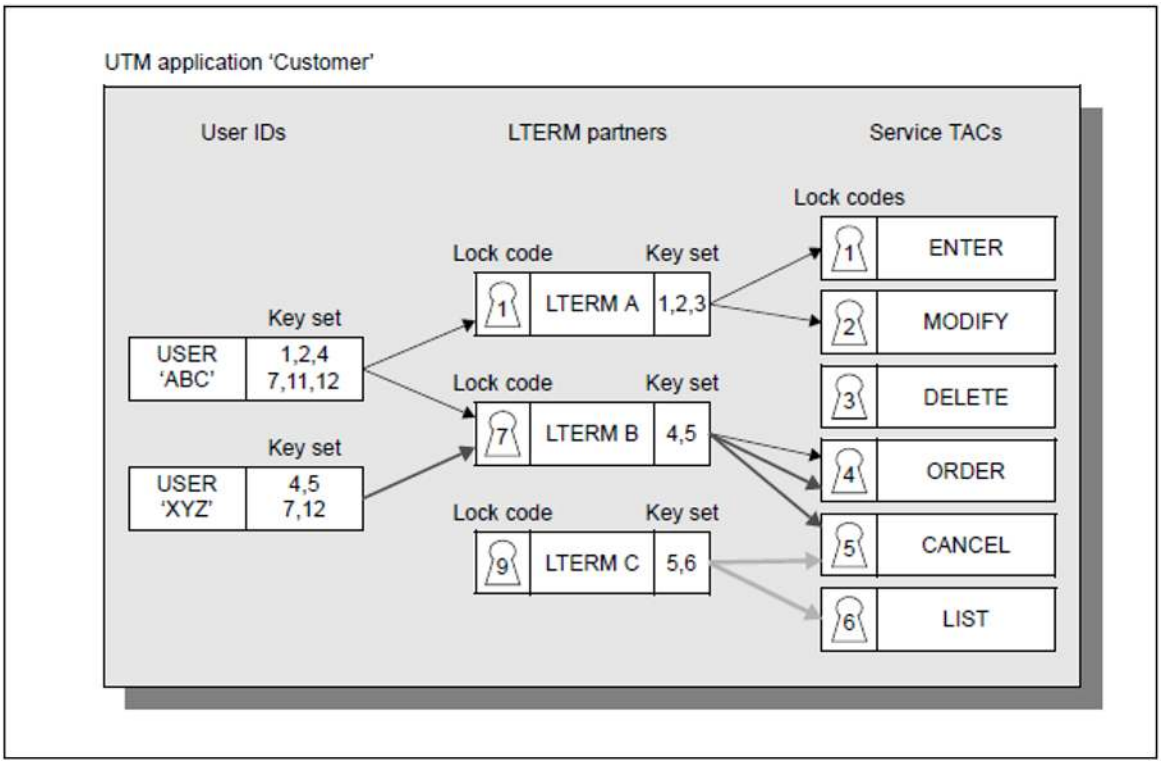


Figure 33: Data access control using the lock/key code concept

A user with the user ID “ABC” wishes to work with the UTM application “Customer”. He must first identify himself to the UTM application by specifying his user ID and, if necessary, a password.

The key set of user ID “ABC” contains the key codes 1, 2, 4, 7, 11 and 12. The user can thus sign on via the client assigned to LTERM partner A (lock code 1), or via the client assigned to LTERM partner B (lock code 4). LTERM partner C is secured with lock code 9. However, since user ID “ABC” does not have the corresponding key code, any attempt to sign on via the client assigned to LTERM partner C will be rejected by the UTM application.

The user can only start a service if **both** his user ID **and** the LTERM partner have the key code that matches the lock code of the corresponding transaction code (service TAC).

LTERM partner A has the key codes 1, 2 and 3. However, since user ID “ABC” is missing key code 3, the user can only start the “ENTER” (lock code 1) and “MODIFY” (lock code 2) services via this LTERM partner.

10.2.2 Access list concept

When the **access list concept** is used, users are grouped together by their roles or functions within the company (porter, clerical worker, personnel worker, head of department, administrator, controller, chief executive officer, etc.). A user can, of course, have more than one role. Each role is mapped to a key code.

- The administrator assigns each user of a UTM application one or more roles (e.g. clerical worker, head of department, etc.).
- The access list is then used to specify which user groups (clerical worker, controller, etc.) have access to the objects (services and TAQ queues) to be protected.
- If you have defined “personnel worker” as role 2, for example, and “chief executive officer” as role 1, you can specify that only these user groups are to have access to the “Personnel” service by assigning the service an access list containing the codes 1 and 2.
- You assign the user a key set that contains all the user’s roles (key codes).

If you use the WinAdmin or WebAdmin administration tool to define access lists and key sets, you can also use meaningful role names instead of numeric codes (UTM converts these symbolic names internally into numeric codes).

LTERM partners can only be protected by means of a lock code. When you use access lists, you should, however, do without the additional data access control of the LTERM partners by means of lock codes (i.e. you should not specify the LOCK operand of the LTERM or TPOOL statement). By assigning a suitable key set to the LTERM partner, you can still ensure that it is only possible to access security-relevant data via particular LTERM partners.

This variant also has the advantage that the key codes in the key set of the LTERM partner do not have to be in the key set of the user. A number of key codes can thus be reserved in an application for access via LTERM partners, for example, because, when a service is accessed, it is enough if only one of the roles of the accessing user is in the corresponding access list, and one of the key codes of the LTERM partner.

If you want to protect a service or a queue by means of an access list, you have to:

- define the access list by means of the KSET statement
- assign the access list to the service or queue by means of the ACCESS-LIST operand of the TAC statement
- define the user-specific key sets by means of the KSET statement
- assign the desired key set to the user by means of the KSET parameter of the USER statement

If you also want to specify that access to security-relevant data is only possible via certain LTERM partners, assign these LTERM partners the appropriate key sets by means of the KSET parameter of the LTERM or TPOOL statement.

A service or TAC queue cannot be accessed unless both the user and the LTERM partner via which the user signed on have at least one role in the access list of the service or queue.

Personnel administration example

In this example, the following roles exist for users and LTERMs:

1: Chief executive officer

2: Clerical worker

3: Telephone operator

10: LTERM with high security level

11: LTERM with normal security level

The application has the services PAYROLL (payroll accounting), PERSDATA (for editing personnel data) and PHONE (for obtaining telephone lists).

This can be illustrated as follows:

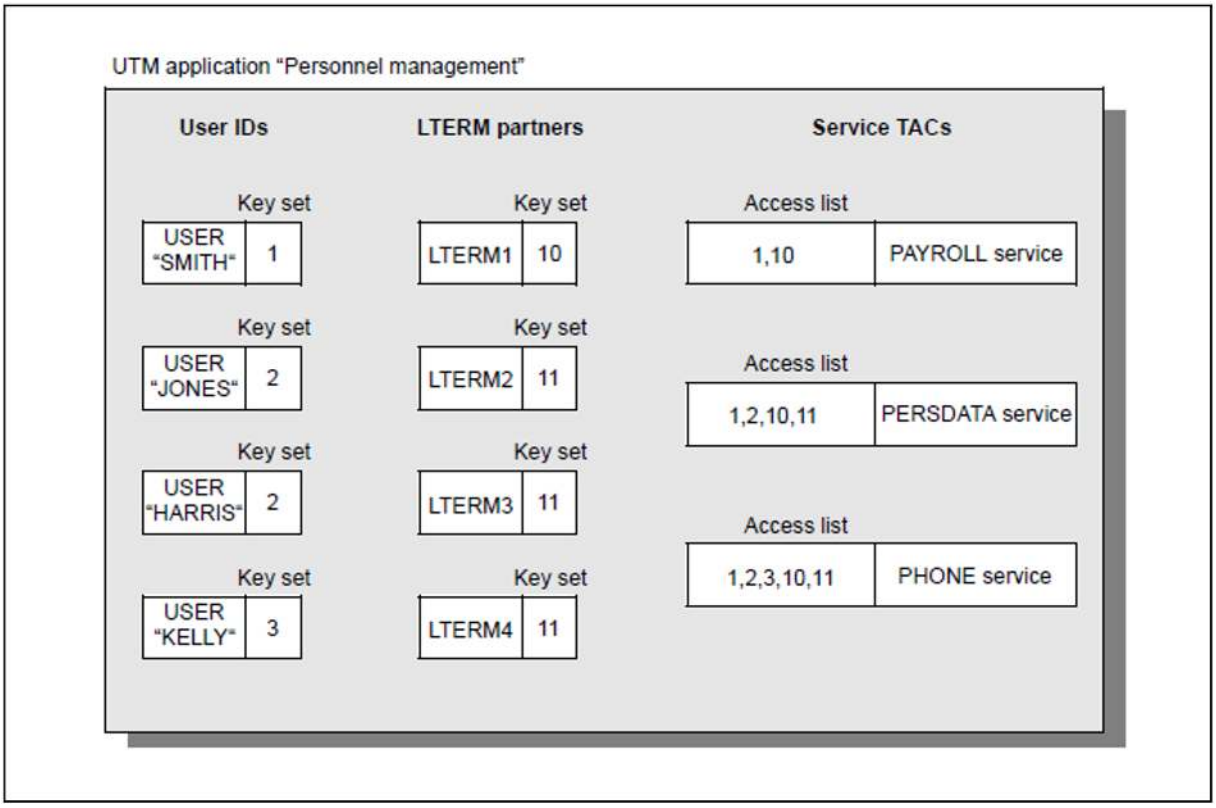


Figure 34: Data access control with the access list concept

A user can only start a service when the key set of the user ID and the key set of the LTERM partner via which the user signs on contain a key contained in the access list of the service:

- The user SMITH is the chief executive officer and the only one permitted to call all the services. To access the PAYROLL service, for security reasons he must sign on via the client that is assigned to the LTERM partner LTERM1.
- The users JONES and HARRIS are personnel clerical workers and can call the PERSDATA and PHONE services (via any LTERM).
- The user KELLY is a telephone operator and can only access the PHONE service.

10.3 System and data access control with distributed processing

The powerful security functions offered by openUTM are also available in cases where several UTM applications interact via server-to-server communication with distributed processing.

System access control through logical access points

UTM applications can only interact with each other if logical access points have been defined in each application for the remote partner applications (from the point of view of the application). These access points are known as (OSI) LPAP partners (see also "System access control (identification and authentication)").

If you are involved in distributed processing via OSI TP, you can also use the UTM user concept globally (see "System access control (identification and authentication)").

Data access control with distributed processing

The openUTM lock/key code concept and the access list concept are also available for distributed processing. The security measures are defined when generating the applications.

- Security measures in the job-submitting application:

  When generating an application, you define which services of a remote partner application are to be accessible. This is achieved by defining a local transaction code (LTAC) for each remote service to be used. As a rule, the application cannot access services for which LTACs have not been defined.

  If you wish to impose additional data access control, you can provide the individual LTACs with lock codes and access lists. A service of the local application can only request a remote service if the user who started the local service has the appropriate key codes.
- Security measures in the partner server application:

  In the partner server application, the job-submitting application is assigned a key set. The service requested by the job-submitting application is started only if this key set contains a key code that matches the lock code or the access list of the requested service.

To access a remote service, therefore, the local application must match the lock code or the access list of the LTAC. The key set assigned by the partner server application to all requests from the job-submitting application must also contain a key code that matches the lock code or the access list defined in the partner server application.

Example: Lock/key code concept for distributed processing

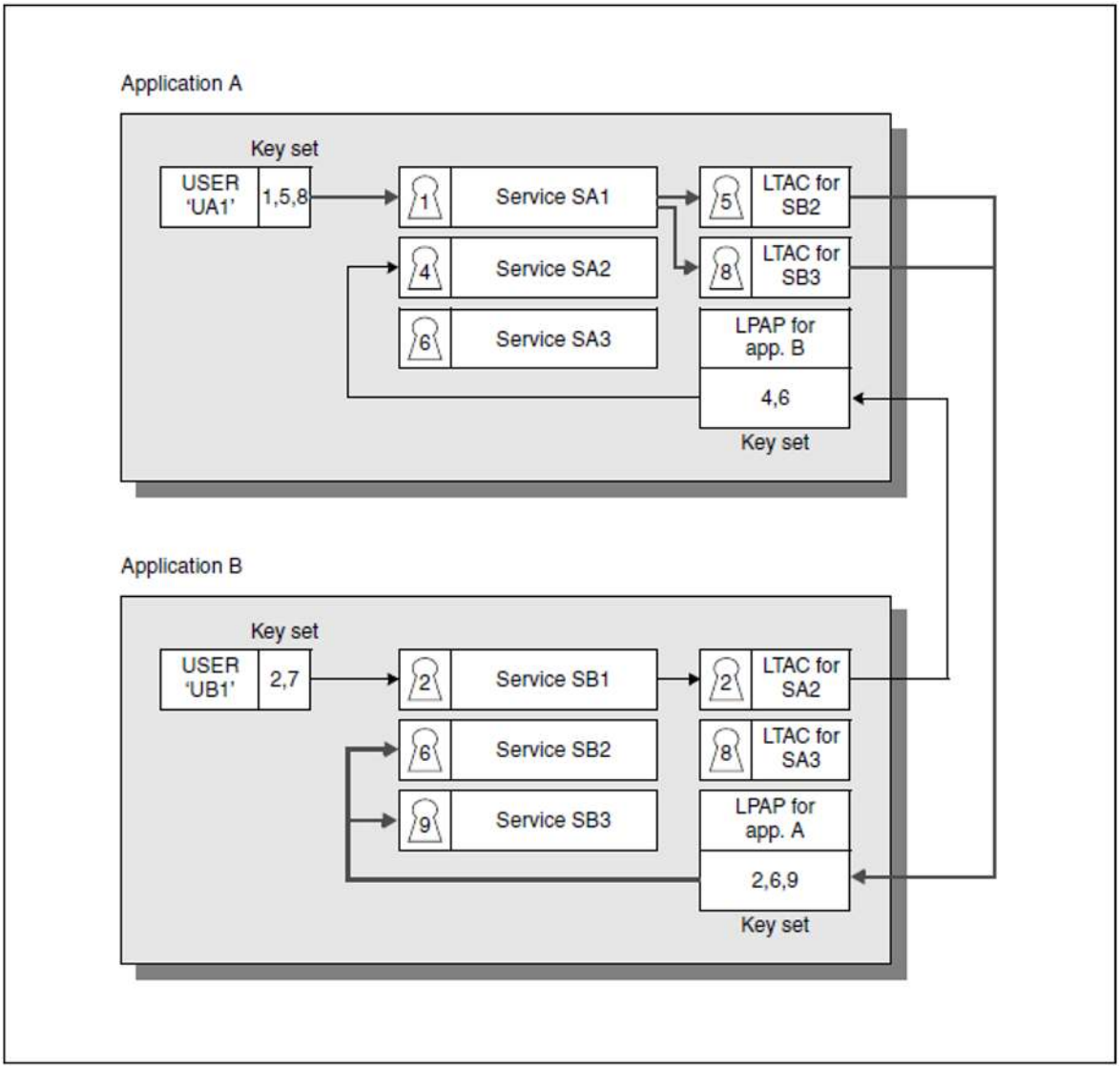


Figure 35: Data access control with the lock/key code concept for distributed processing

In the example shown in figure 35, server-to-server communication between application A and application B is possible, since an LPAP partner has been generated in each application for the respective remote application. In application A, LTACs have been generated only for the remote services SB2 and SB3. Service SB1 is therefore not available to application A. A user that signs on to application A under the user ID “UA1” has the key code 1, and can therefore access service SA1 in application A.

Since the user also has the appropriate key codes for the two LTACs, service SA1 can request services SB2 and SB3 via these LTACs. The key set assigned by application B to all requests from application A contains appropriate key codes for services SB2 and SB3. All requirements are thus fulfilled in the configuration of both application A and application B: service SA1 started under user ID UA1 in application A can access services SB2 and SB3 of application B.

Example: Access list concept for distributed processing

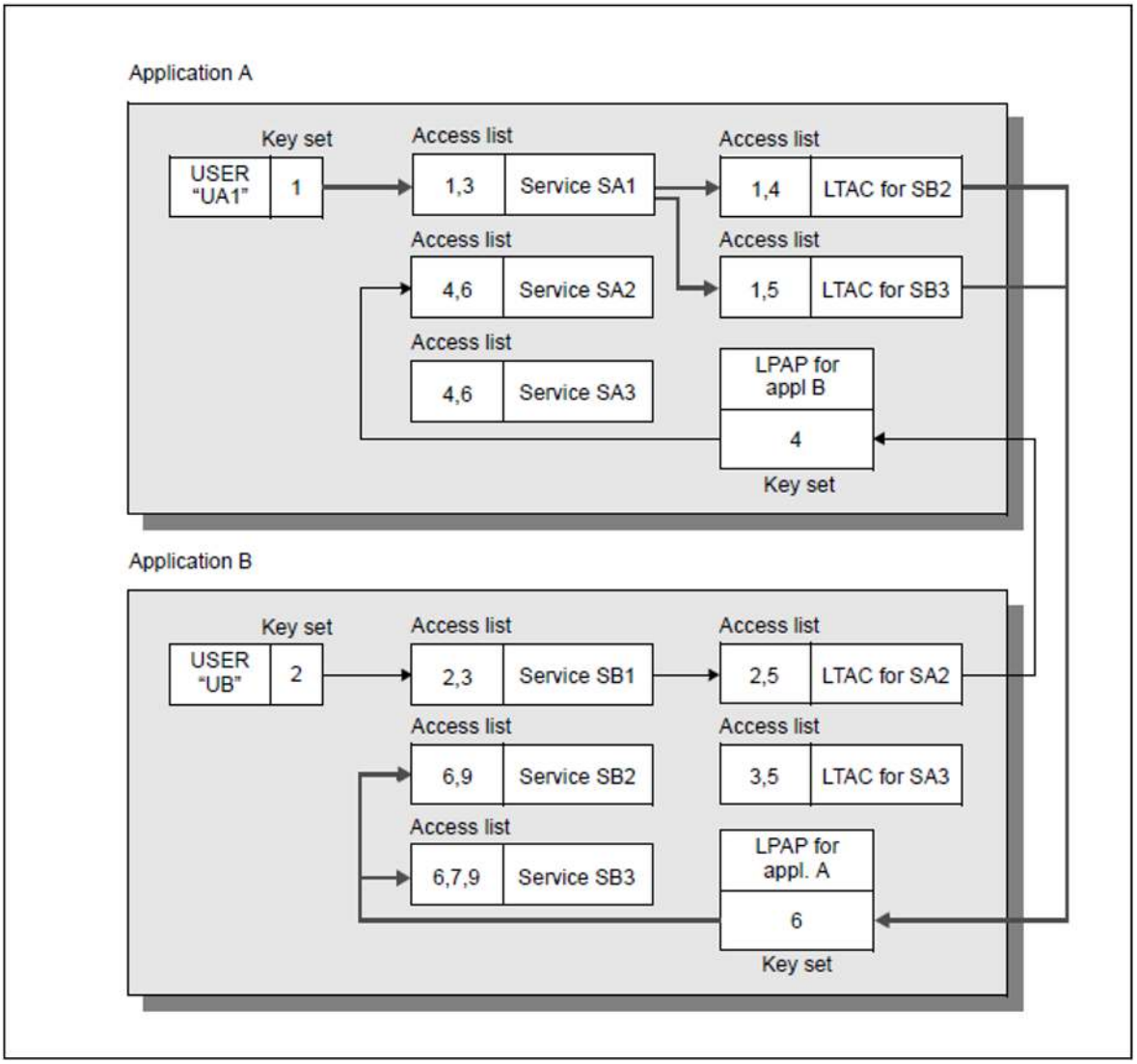


Figure 36: Data access control with the access list concept for distributed processing

In the example shown in figure 36, server-to-server communication between application A and application B is possible because in both of the applications there is an LPAP partner generated for the remote application. In application A, LTACs are generated only for the remote services SB2 and SB3. The service SB1 thus cannot be used from application A.

A user who signs on to application A under the user ID “UA1” has the role “1”. This allows the user to use the service SA1 in application A. Because the role “1” also permits access to the two LTACs, the service SA1 can request the remote services SB2 and SB3 via these LTACs. In application B, the LPAP has the key “6” for application A, which permits the services SB2 and SB3 to be called.

All the prerequisites are thus fulfilled to allow the service SA1 started under the user ID UA1 in application A to access the services SB2 and SB3 of application B. The same applies by analogy when the user UB1 wants to access the service SA2 in application A via the service SB1.

10.4 Encryption

Clients often access UTM services via open networks. This can allow unauthorized persons to read data transmitted on the line and pick out passwords for UTM users or read sensitive user data. openUTM supports the encryption of passwords and user data in connections to UPIC clients to prevent this from happening.

For encryption openUTM uses either a combination of RSA methods (named after the authors Rivest, Shamir and Adleman) and AES-CBC (Advanced Encryption Standard Cipher Block Chaining Mode) or DES methods, or a combination of Ephemeral Elliptic Curve Diffie-Hellman methods (ECDHE) and AES-GCM methods (Advanced EncryptionStandard Gallois Counter Mode). The second combination is more state-of-the-art and offers increased security compared to the first combination, but is currently only supported by openUTM for Linux - UNIX - Windows platforms.

The AES key is generated by the UPIC client each time a client-server connection is established.

The RSA key pair is generated by openUTM and consists of a *public key* and a *private key*. Several key pairs with different key lengths can be generated in an application. This enables several encryption levels to be implemented.

Use of the RSA-AES-CBC encryption method

Data is encrypted and decrypted in the following steps.

- When the connection is established, the public RSA key associated with the generated encryption level is passed to the UPIC client. If no encryption level has been generated (NONE), then the longest currently available RSA key is passed.
- The UPIC client generates a connection-specific AES key. This is encrypted using the public key and sent to the UTM application.
- Once the UTM application has decrypted the AES key correctly using the RSA key, the client sends the encrypted data.
- openUTM decrypts the data using the AES key.
- All other data exchanged between client and openUTM on this connection is also encrypted with the AES key.

Use of the ECDHE-RSA-AES-GCM encryption method (only for Linux, Unix and Windows systems)

The combination ECDHE-RSA-AES-GCM uses the elliptic curve based Ephemeral Diffie-Hellman method to agree the AES session key. Each side generates a Diffie-Hellman key pair, transmits the public part of its key pair to the partner and generates the common AES session key from its private key and the public key of the partner. This means that in this procedure the AES session key is not transferred on the data connection.

The server also signs its private key with the private RSA key of the UTM application. In this way, the client can verify that the sent public Diffie-Hellman key of the server really comes from the UTM application. Again, as described above, to prevent man-in-the-middle attacks, it is necessary that the public RSA key is additionally made known to the client in a separate way.

The Ephemeral Diffie-Hellman method offers the user Perfect Forward Secrecy, this means that even recorded data cannot even be decrypted if the Long-Term key (RSA key) is compromised in the future.

The AES-GCM procedure is used to encrypt user data. Compared to AES-CBC, this method has the advantage that it supports Authenticated Encryption with Associated Data (AEAD), in which the encrypted message and the other protocol parts of the message are protected against changes by a Message Authentication Code (MAC).

It is also possible to read out the public RSA key of an UTM application via the administration interface and to have it securely stored locally on the UPIC clients. In this case the client can check when a connection is established whether the public key received matches the locally stored key (to prevent a „man-in-the-middle“ problem).

For security reasons, the RSA key pair of a UTM application should be replaced with a new pair at regular intervals (by means of programmed administration or using WinAdmin/WebAdmin). Administration facilities can be used to activate, deactivate and delete RSA keys. It is therefore possible to distribute a newly generated RSA key to all UPIC clients, to activate it and then to deactivate or delete the old key pair.

Encryption can be used in openUTM to control the access to the *system* by clients as well as the access to certain *services*.

System access control

You can specify in the UTM configuration for each client and each group of clients if and to what extent the messages and passwords are to be encrypted. There are three cases in this regard:

1. The client must always perform encryption, otherwise it will not gain access to the UTM application. Several encryption levels can be defined. The client must perform encryption to at least the requisite encryption level.
2. The client is allowed access without encryption, but it must encrypt when a service explicitly demands it to encrypt (see below under “Access protection”).
3. The client is a *trusted* client and does not need to encrypt.

Access protection

openUTM can protect individual services. A client may only access such services when it is a trusted client or when it can perform encryption to at least the requisite encryption level.

If a client that is not a trusted client wants to access a service with this kind of protection, then it may be required to send an encrypted input message. openUTM sends the output message back in encrypted form even if the client has started the service without an input message or the service was started through service chaining.

Information on encryption for clients and service can be found in the openUTM manual “Generating Applications” under the keyword ENCRYPTION-LEVEL and in the openUTM manual “Administering Applications” under the keyword KC_ENCRYPT.

10.5 Security functions of external resource managers

If your UTM applications interact with external resource managers, e.g. database systems, you can also use the security mechanisms of these resource managers. The openUTM security mechanisms are kept completely separate from those of the database system, and there is no delegation, i.e. the security functions do not “pass through” individual users. From the point of view of the database, the UTM application functions as a user.

This has a number of advantages:

- clear differentiation between the “Application” and “Data storage” tiers
- data access control on the service level rather than the data level
- no unnecessary redundancies in the authorization profiles, thereby reducing the effort involved in administration and eliminating inconsistencies (e.g. anomalies in modifications)
- faster access times and more efficient utilization of resources in the database system (administration tables, caches)

You can alternatively specify the access data for Oracle databases interfaced to openUTM as an XA Resource Manager in the start parameters in plain text form or define them in the generation of the UTM application. For security reasons, it is recommended to specify the access data only in the generation.

In addition, the access data can be modified by means of administration.

11 High availability with standalone UTM applications

One of the most important demands made of business-critical applications is that they should be available 24 hours a day, 365 days a year. System failures, and thus application failures, can be very costly, not to mention the damage to the image of the company running the application.

In addition to the computer configuration, the operating system is central to ensuring high availability. No operating system can provide 100% availability. But BS2000 systems, Unix, Linux and the Windows systems do offer a large number of functions that go a long way toward achieving this.

openUTM supports the high availability offered by these operating systems by means of global transaction management and restart capability. openUTM and the operating system thus complement each other ideally and together ensure practically fault-tolerant operation of the application.

11.1 High availability on BS2000 systems

The high-availability functions offered by BS2000 systems include the dynamic reconfiguration of the hardware and software, dynamic network reconfiguration, time switching with no interruptions, and the minimization of downtimes.

11.2 High availability on Unix and Linux systems

Failure of computer systems, hardware or software can result in high costs in some applications. Preventive measures are therefore required as a prerequisite for smooth operation:

- the use of cluster configurations
- special disk peripherals such as RAID systems
- software that identifies faults and enables you to switch quickly from defective hardware to backup systems
- distribution of a failover cluster on remote computing centers

These measures, together with openUTM, can be utilized to their full potential because openUTM offers global transaction management and comprehensive restart capabilities right up through the client. This makes openUTM especially suited for use in clusters because processing can be continued on a different system after a restart without losing any data or allowing inconsistencies to arise.

Linux systems can, for example, be used as clusters with PRIMECLUSTER RMS. This software detects failures, performs auto recovery and/or automatically switches from defective systems to intact systems (auto-switchover). UTM applications that have been prepared accordingly can continue operation on the intact system immediately or after a system restart.

More information on this topic can be found e.g. in the manuals for PRIMECLUSTER on Linux.

11.3 High availability on Windows systems

Windows systems support high availability by means of:

- the use of cluster configurations
- special disk peripherals such as RAID systems
- software that identifies faults and enables you to switch quickly from defective hardware to backup systems

These measures, together with openUTM, can be utilized to their full potential because openUTM offers global transaction management and comprehensive restart capabilities right up to the client. This makes openUTM especially suited for use in clusters because processing can be continued on a different system after a restart without losing any data or allowing inconsistencies to arise.

Windows systems using PRIMERGY hardware can be used as follows as a cluster:

- Symmetric cluster with the Microsoft Cluster Server:A Microsoft Cluster Server configuration currently consists of two independent Windows-based servers that are connected via a high-speed connection and that presents itself to the user as a single server system. The clients in the network can access all resources of the cluster transparently. If a server crashes, then the other servers automatically take over its function. Both servers can be run in mission-critical mode, i.e. they work productively on their own applications and also monitor each other.
- Asymmetric cluster with the PRIMERGY ServerShield and SCSI switch:The cluster consists of primary and secondary servers. The primary server processes the business critical tasks by default while the secondary server assumes less critical tasks. The secondary server monitors the primary server. If it detects an error in the primary server, then it properly terminates its own applications and assumes the data of the primary server. The new start of the secondary server with the data and applications of the primary server quickly restores the availability of the overall system.

12 High availability and load distribution with UTM cluster applications on Linux, Unix or Windows systems

Downtimes are unacceptable wherever applications are deployed in mission-critical corporate areas. High availability and the ability to distribute load automatically at peak load times are the highest priorities.

openUTM supports high availability through the UTM cluster functionality on Linux, Unix or Windows systems. In addition, openUTM also permits load distribution by means of LPAP bundles for LU6.1 and OSI TP connections as well as the UPIC load distributor for UPIC clients.

For basic information on the openUTM cluster functions, refer to section “UTM cluster application on Linux, Unix or Windows systems”.

12.1 High availability with UTM cluster applications

The figure below provides an overview of the central high availability functions provided by a UTM cluster application::

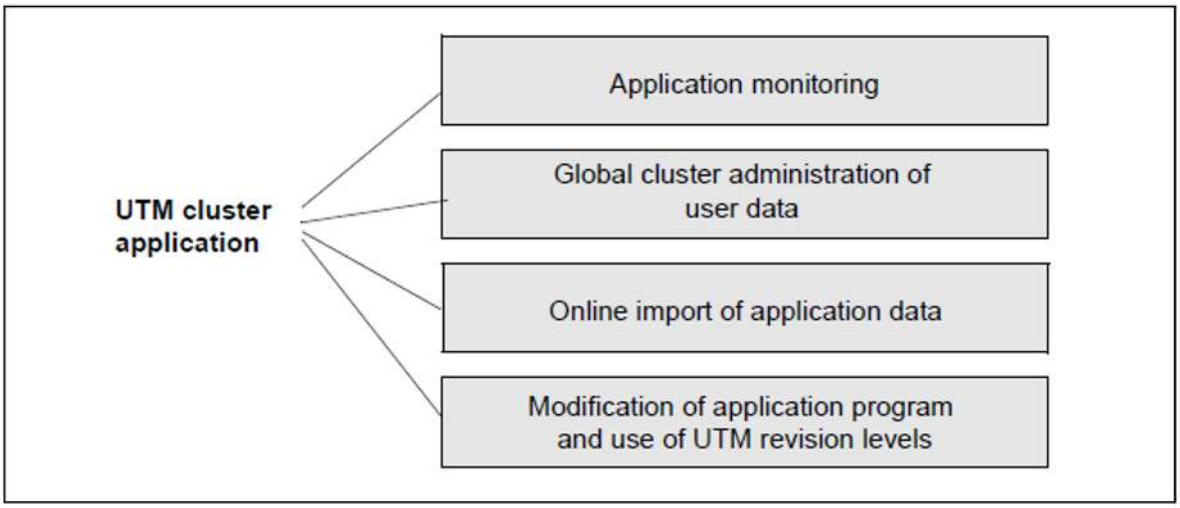


Figure 37: Central high availability functions of a UTM cluster application

Application monitoring and measures on failure detection

It is possible to monitor UTM cluster applications without the need for additional software. The node applications monitor each other. If a node application fails, it is possible to trigger follow-up actions.

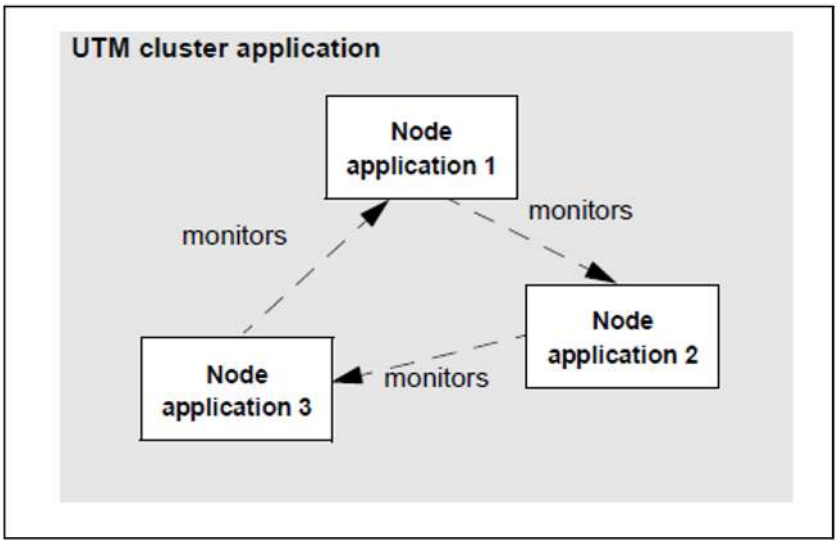


Figure 38: Circular monitoring between three node applications

When a node application is started, it is decided dynamically what other node application is to be monitored by this note application and what other node application monitors this node application. These monitoring relationships are entered in the cluster configuration file. When the application is terminated, the relationships are canceled. After a monitoring relationship has been set up, the existence of the monitored node application is checked at a specified interval that can be generated.

The availability of a node application is monitored within a two-stage process in which, at any time, the next stage is initiated if the stage that precedes it cannot exclude the possibility that the monitored application has failed.

- The first stage in this monitoring process is performed through the exchange of messages.
- The second stage consists of an attempt to access the KDCFILE of the monitored application in order to check whether the monitored application is still active.

In the event that failure of a node application is detected, you can use a procedure or script to initiate follow-up actions. The content of the procedure or script depends on the actions and the platform on which the node applications are running and is defined by the user.

For more detailed information on application monitoring in the cluster, see the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”

Global administration of application data in the cluster

Some application data such as the global memory areas GSSB and ULS and the service-specific data relating to users is administered at global cluster level. This means:

- GSSB and ULS can be used globally in the cluster. As a result, the current contents of GSSB and ULS are always available in every node application. Changes made to these areas by a node application are immediately visible in all the other node applications.
- Service restarts are node-independent. Following the normal termination of a node application, users can continue an open dialog service at another node application provided that the service in question is not a node bound service. Such users can continue working without delay and do not have to wait until the terminated node application becomes available again.

The application data is administered in UTM cluster files which can be accessed by all the node applications. A brief description of these files can be found in the section “UTM cluster files”.

For more detailed information on UTM cluster files in UTM cluster applications, see the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

Special characteristics of the LU.6.1 link

The LU6.1 protocol uses so-called sessions to perform communication. The sessions have two-part names that must be known at both partners. The session name is therefore nodespecific. Since all node applications are generated in exactly the same way in UTM cluster applications, all the session names are present in a node application, including the session names used for the other node applications. To permit the UTM application to select an appropriate session when establishing a connection to a partner application, the logical name of the node application must also be assigned to each of the sessions on generation.

For more detailed information on generating an LU6.1 link for a UTM cluster application, see openUTM manual “Generating Applications” under "KDCDEF control statements - sections CLUSTER-NODE and LSES" as well as under "Distributed processing via the LU6.1 protocol".

Online import of application data

Following the normal termination of a node application, another running node application can import messages to (OSI) LPAPs, LTERMs, asynchronous TACs or TAC queues and open asynchronous conversations from the terminated node application. The imported data is deleted in the terminated node application.

Online imports must be initiated at administration level.

For more detailed information on the online import of application data in the cluster, refer to openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

Online application update

The use of UTM cluster applications permits genuine 7x24 operation. The ability to modify the (static) configuration is an important aspect of high availability.

It is possible, while a UTM cluster application is running, to start up a new version of the application program, make changes to the configuration or deploy a new UTM revision level.

The operations necessary to do this are described below.

New application program

If you want to add new application programs to a UTM cluster application or modify existing programs, you do not have to shut down the entire UTM cluster application to do so.

In addition to the possibilities for changing programs during operation – for example, by adding programs at the administration level, rebinding the load module, exchanging the load module – it is also possible to make changes to the application program during operation. When you do this, you must briefly shut down the corresponding node applications in sequence. For example, in this way you can add a new load module to the application or add programs in excess of the reserved space available (see openUTM manual “Generating Applications”, RESERVE statement).

After you have added the application program, you can restart the relevant node application with the new application program.

New configuration

If you want to make changes to the configuration of a UTM cluster application that are not possible using the dynamic administration capabilities, it is not always necessary to shut down the entire UTM cluster application.

To do this, you must regenerate your UTM cluster application. After generation, you must take over the KDCFILE for each node application.

For a detailed description of the procedure and the actions required for the individual node applications, refer to the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

UTM revision levels

You can deploy UTM revision levels during system operation, i.e. some of the node applications can continue to run while the revision level is being implemented in the other node applications.

To do this, you must shut down the node applications one after the other and then restart them with the new revision level.

For more detailed information on the online updating of UTM cluster applications, refer to the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”

Node recovery on another node computer

If a node application terminates abnormally due to a computer failure and cannot be restarted rapidly on the failed computer then node recovery can be performed for this node application on any other node computer in the UTM cluster.

This makes it possible to eliminate the consequences of the abnormal termination of a node application, for example by releasing the ULS and GSSB memory areas that are global to the cluster and are locked on the abnormal termination of the node application or by signing off users who were signed on exclusively at the node application at the time of termination.

Following node recovery, node or cluster updates and online import operations are possible again.

openUTM supports node recoveries even when interacting with databases. For this to be possible, the associated database must provide the required functions. For details, see the openUTM Release Notice.

For more detailed information on the prerequisites for and configuration of node recovery operations, see the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”

12.2 Load distribution

In the case of distributed processing in a cluster, automatic load distribution is possible using the LU6.1 and OSI TP protocols. It is also possible to distribute the load at UPIC clients in order to balance the incoming load.

12.2.1 Load distribution for distributed processing

For communications between a standalone UTM application and a UTM cluster application via the OSI TP or LU6.1 protocol, openUTM offers LPAP bundle functionality.

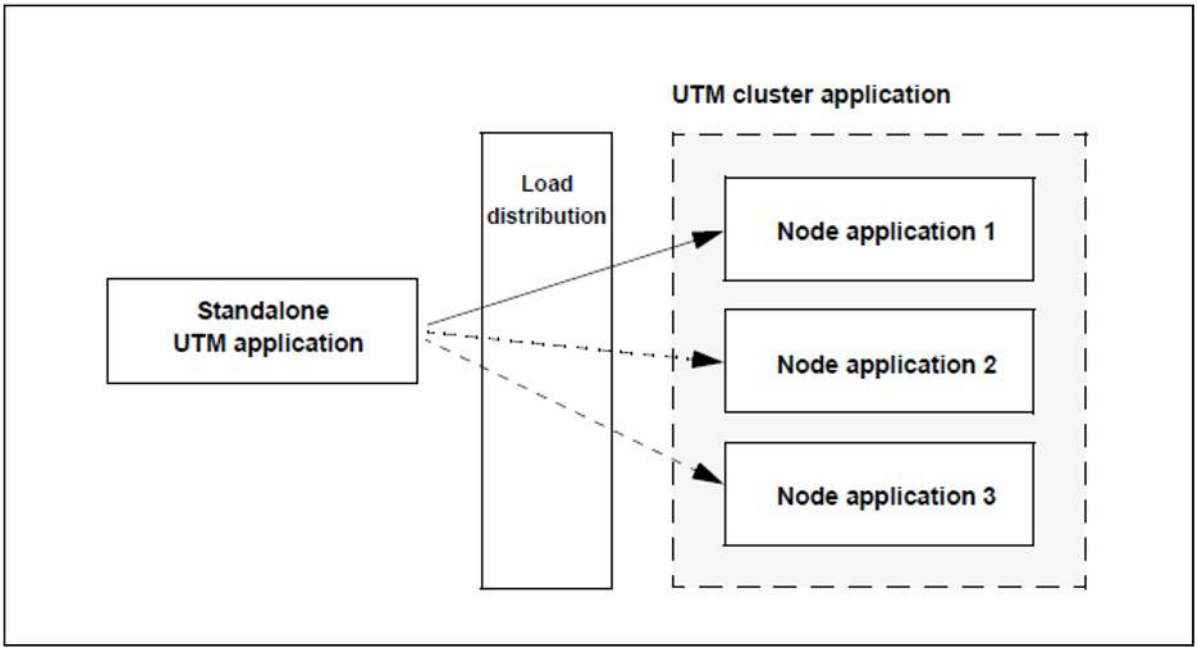


Figure 39: Communication between a standalone UTM application and UTM cluster application with load distribution

Thanks to this functionality, jobs sent by the standalone application to the UTM cluster application are distributed to the available node applications.

The LPAP bundles must be generated in the standalone UTM application.

For more detailed information on automatic load distribution via (OSI) LPAP bundles, refer to openUTM manual “Generating Applications”.

Load distribution between two UTM cluster applications

The LPAP bundle concept is also applicable in situations in which UTM cluster applications are present on both sides. If only one of the two UTM cluster applications sends requests to the other party then it is only necessary to generate LPAP bundles in this UTM cluster application. If both parties send requests to one another then LPAP bundles are required on both sides.

12.2.2 Load distribution at UPIC clients

Jobs submitted by UPIC clients to the UTM cluster application are distributed across the individual node applications of the UTM cluster application. One node application with which the next UPIC communication is performed is randomly selected from a list of node applications.

For more detailed information on load distribution in UPIC clients, refer to the manual „openUTM-Client for the UPIC Carrier System”.

12.2.3 Load distribution with Oracle® RAC

Optimum load distribution to the Oracle® RAC nodes can be achieved in a UTM cluster application.

For more detailed information on load distribution with Oracle® RAC, refer to the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

13 Fault tolerance and the restart function

openUTM not only protects your data and applications through the sophisticated mechanisms for system and data access control described in the chapter “Security functions”, but also guarantees security and consistency, even in the event of errors or faults. The effect of errors in the application program remains local, and user errors are intercepted. Internal validation routines detect system errors or inconsistencies and respond automatically.

Even if the server or operating system crashes, openUTM guarantees that no data will be lost. If a fatal error is detected, the UTM application automatically terminates before any damage can be done. Universal restart functions also ensure that you can continue working immediately with consistent data as soon as all users have been restarted. See section “The openUTM restart functions” for more information.

This makes openUTM in **cluster-capable**, i.e. processing can be continued on a different system after the restart (see also chapter “High availability with standalone UTM applications”).

13.1 Limiting program unit and formatting errors

openUTM limits the effect of errors in program units or formatting errors. Other transactions and the application as a whole are not affected.

openUTM provides the following functions:

- In the case of minor errors or faults, openUTM outputs return codes which describe the error. This gives the relevant program unit the opportunity to react accordingly to the error or fault.
- Since UTM applications work with several processes in parallel, a serious error in an application program results at worst in termination of the relevant process. UTM applications therefore do not have a single point of failure. A terminated process is automatically restarted. Since the operation for rolling back all open transactions - including those of external resource managers - is fully coordinated, data consistency is maintained across all applications.
- A local storage area is automatically assigned to each process of a UTM application and contains current system data for a job or program run. The data area can only be accessed by the local process, and access is controlled by means of a UTM system code, i.e. the processes of a UTM application are isolated from each other.
- If dialog services are aborted due to a serious error, openUTM informs the client of the error. The client can then start further services. If an asynchronous service is aborted, the job is removed from the queue so that subsequent jobs can be started. Using acknowledgment jobs, it is also possible to inform the job submitter of errors.
- If errors occur when formatting messages, openUTM detects these and reacts accordingly:
  - Dialog services are aborted. The relevant client is informed by means of a message and can then start other services.
  - After aborting an asynchronous service, openUTM removes the job from the queue. Subsequent jobs can then be started unhindered.
- Each UTM program unit must end with a special UTM call. If you are working with the KDCS program interface, this is the PEND call. This ensures, amongst other things, that local storage areas are released. If an error has not been intercepted in the program unit (with PEND ER in KDCS), openUTM issues a PEND ER call internally: in any case, the process is shut down gracefully and a dump is automatically created.
- The KDCS storage areas KBPROG and SPAB can be defined at the end of a dialog step using any character defined during generation. openUTM also checks whether a program unit requires a larger storage area (KBPROG / SPAB) than requested during generation.

13.2 Automatic checks

Detecting user errors

If a terminal user enters an illegal user command, openUTM responds with an appropriate message.

If a terminal user presses illegal keys when filling out an input format, openUTM detects this and outputs the relevant input format again. The terminal user is thus given the opportunity to correct the erroneous input.

Consistency checks when starting the application

When starting the application, openUTM performs automatic consistency checks. For instance, if an attempt is made to start an application using components from different openUTM versions, or if the configuration files (KDCFILE and possibly also UTM cluster files) do not match or are inconsistent, openUTM aborts the startup process and outputs a message describing the nature of the error.

Internal validation routines

The UTM code contains a series of internal validation routines. If errors or inconsistencies are detected which may corrupt the application data, openUTM immediately terminates the UTM application, closes all open files, and creates a specific UTM dump for each application process. The UTM application can then be restarted with logically consistent data. During an automatic restart, each client is informed of the processing state of its own particular jobs.

13.3 Faults or crashes in local resources

openUTM guarantees that no data will be lost, even if the local resources fail. For instance, it offers protection against the problems described below.

Operating system error

If an operating system function informs openUTM that an error has occurred, then this error is handled as an internal openUTM error, i.e. before any damage can be done, openUTM immediately terminates the application with an appropriate error code.

Disk failure

To enhance data protection, it is possible to mirror the KDCFILE on different drives (hot standby). This may result in a very slight increase in input/output times (by no means doubled), but the effect on performance is negligible.

Hardware errors in terminals

If a terminal fails, the user can simply move to another terminal, sign on there under his/her user ID, and resume the service started.

Even if the application operates without user IDs, the terminal user can immediately continue working on another terminal if his/her terminal breaks down. In this case, however, administration must assign the user another terminal.

If a defective terminal cannot receive asynchronous messages, it is possible to assign an intact terminal to the relevant logical access point (LTERM partner) by administration. The asynchronous messages are then output on this terminal.

Network failures or serious network faults

In the event of a network failure, the relevant network connections to openUTM are shut down.

Loss of terminal connections

When the (sub)network has been restored, terminal users can sign back on to the UTM application and continue working.

Loss of printer connections

A network failure or a serious network fault can result in the loss of printer connections. This also occurs if a logical print acknowledgment is not returned. openUTM uses logical print acknowledgments to monitor the printing of messages. If a requested print acknowledgment
is not returned within a defined period, openUTM shuts down the connection to this printer.

In all of these cases, openUTM automatically attempts to reestablish the connection. These attempts are repeated at defined intervals. If the connection still cannot be reestablished, the administrator can assign another printer. If you have a printer pool, the messages are automatically output to another printer.

After the connection has been reestablished, the entire print process is repeated in the following situations:

- The message to be printed was sent in full, but a print acknowledgment was not returned within the defined period.
- Several parts of a message were sent, but the entire message was not printed in full.

To indicate the possibility of duplicate messages, openUTM sends an appropriate message with each output attempt.

Errors that do not result in a loss of connection

Errors can occur when inputting or outputting messages. These may be due to hardware errors in terminals or printers, or network faults. During input, such errors are detected by the formatting system. In this case, openUTM outputs the last output message again, so that the terminal user can repeat his/her input. If the terminal user identifies faulty output messages (e.g. smudge characters), he/she can output the last message again (KDCDISP user command).

If a printer is found to be faulty before openUTM has finished printing the entire print message, then openUTM will clear down the connection. The message is output again as soon as the connection has been reestablished.

13.4 Abnormal termination of a UTM application

As described in the preceding sections, openUTM must abort a UTM application to protect programs and data (abnormal termination) in certain exceptional situations. The application is then automatically restarted by openUTM during the next start (see section “The openUTM restart functions”).

When terminating the UTM application, openUTM performs the following tasks if possible:

- it shuts down connections to external resource managers (e.g. database systems)
- it closes all files
- it creates a UTM dump for each process of the UTM application for error analysis
- it terminates all processes

Following an abnormal termination of a UTM application, no attempt is made to restore the consistency of the KDCFILE, which contains all data required to run a UTM application. For security reasons, this should be postponed until the application is restarted.

13.5 The openUTM restart functions

The openUTM restart functions ensure that services and jobs interrupted by errors, faults or crashes can be resumed or repeated immediately with consistent data. It does not matter if individual services (e.g. due to the loss of a connection) or entire UTM applications (e.g. due to a server crash) have been aborted.

A requirement for this to function is that openUTM enters records containing restart information in the KDCFILE while the application is running (transaction logging). openUTM offers two function variants with different restart behavior.

13.5.1 The UTM-S and UTM-F variants

openUTM offers the two variants UTM-S and UTM-F for the operation of an application. These two variants can be characterized as follows.

- **UTM-S** (UTM-Secure): a high level of security through universal restart functions
- **UTM-F** (UTM-Fast): high performance with limited restart functions

UTM-S works with comprehensive transaction logging of all user and administration data. UTM-F, on the other hand, only executes transaction logging for administration data and therefore requires fewer I/Os. For this reason, UTM-F has better performance, but does not have complete restart capability. The two variants barely differ with respect to the administration data. Details can be found in the openUTM manual “Administering Applications” in the sections describing the corresponding administration calls.

You specify which variant a UTM application will use during generation of the UTM application.

The two variants function in the same manner during operation. UTM-F also executes transaction processing according to the “all or nothing” principle and guarantees the consistency of the data when used in conjunction with *one* database system.

Differences between the two variants only become apparent after the end of the application run (normal termination or an application crash):

In the case of standalone applications, for example, UTM-F does not save any user data for the following restart (application warm start). In UTM cluster applications, UTM-F does not save the user data until the user signs off.

The following sections describe in more detail how the two variants behave during a restart.

13.5.2 Restart with UTM-S

Following abnormal termination of a UTM-S application (UTM-Secure), openUTM offers enhanced, high-speed restart (= warm start) functions. All open transactions are rolled back to a consistent state, interrupted services are restarted, and users can continue working with the screen belonging to this consistent state (screen restart).

For this purpose, openUTM enters a record containing restart information in the KDCFILE for each transaction during operation of the UTM-S application.

Consistency for interrupted transactions

However, openUTM does not simply roll back all open transactions. Instead it proceeds as follows:

- Transactions that were undergoing end-of-transaction processing at the time of the failure are handled differently by openUTM, depending on whether or not the transactions of external resource managers were complete when the error occurred.
  - If the transactions of the resource managers were **not complete** and were rolled back when the resource manager was restarted, openUTM also rolls back the UTM transaction.
  - If the transactions of the resource managers were **complete**, openUTM also completes all UTM transactions open at the time of the failure (commit).

  This also applies to the case of distributed transactions.
- All transactions that had not yet undergone end-of-transaction processing at the time of the failure are rolled back.

Restarting dialog services

Interrupted dialog services may be resumed under certain circumstances after an application restart (service restart).

- If a user has an open service and has signed on via a sign-on service, then the sign-on service decides if a service restart is to be performed or the service is to be terminated abnormally.
- If a user has an open service and has signed on via a client program, then a service restart must be explicitly requested by the client program, otherwise openUTM terminates the service abnormally.
- If a user has an open service and has signed on via a terminal, then openUTM always performs a service restart.

The response of openUTM depends on whether or not the last action was closed with a synchronization point. When a service is restarted, it is always rolled back to the last synchronization point.

Detailed information on the restart procedure when connecting client programs can be found in the openUTM-Client manuals.

Restarting background and output jobs

The openUTM transaction-oriented queuing mechanism ensures that all queues containing background or output jobs are retained following a failure that openUTM has accepted for processing and which have not yet been completely processed.

openUTM operates as follows during a restart:

- Background jobs whose processing has not yet been started or that have not yet reached a synchronization point are restarted after the restart.

- Background jobs that have already reached a synchronization point will be processed starting at this synchronization point.
- All output jobs whose processing has not yet been started or that have not yet been completely processed at the time of the crash are available for output after the restart.
- All messages from service-controlled queues that have not yet been read in completed transactions are available for reading again after restart.

13.5.3 Restart with UTM-F

When a UTM-F application is running, openUTM saves a large portion of the administrative changes in the administration file for the restart. This includes the changed passwords, new users and the locking of TACs, for example. See also the openUTM manual “Administering Applications”. These changes are also made available after a new start of the UTM-F application.

In contrast, in UTM-F

- No user data from standalone applications is stored in the KDCFILE. Correspondingly, all user data is “forgotten” in UTM-F during a normal as well as during an abnormal termination of the application. This includes, for example, secondary storage areas such as the GSSBs, information on open dialogs or background jobs.
- In UTM cluster applications, the user’s service data is not saved until the user signs off in the cluster page pool. If, for example, a user is signed on at a node application that terminates abnormally then his or her user data is “forgotten”. In contrast, in the case of UTM-F cluster applications, GSSB and ULS are saved in UTM cluster files when the transaction is terminated - unlike in the case of standalone applications.

For this reason, the consistency of the data in a UTM-F application after a restart can only be guaranteed when only local transactions are used and when all user data is maintained in a single database.

13.6 Error handling for distributed processing

The following error situations can occur with distributed processing:

- errors in one of the applications
- loss of connections between applications
- termination of one of the applications involved in open transactions

openUTM reacts to these errors with messages indicating the cause of the error. In the case of errors within an application, errors codes and usually a dump are created. The partner application is informed of the errors as soon as possible.

The roll-back and restart functions behave differently, depending on whether you are working with global transaction management or with independent local transactions.

13.6.1 Roll-back and restart functions with global transaction management

Distributed processing with global transaction management can be done via the LU6.1 or OSI TP protocols. openUTM always works with global transaction management for LU6.1 while global transaction management is only valid for OSI TP if the “Commit” function unit is selected.

Rolling back distributed transactions

There are a number of reasons why openUTM will roll back a distributed transaction, e.g. violation of programming rules, loss of connection, timeout when monitoring response times, termination of an application, or in certain situations after the failover of a database with open transactions, see "Coordinating with databases and resource managers" as well.

Depending on the reason for the roll-back, either the job-submitting and/or the job-receiving services are terminated because resumption was not possible or practical, or the jobsubmitting and the job-receiving service are rolled back to the last synchronization point and communication is restarted (service restart).

Detailed information on this can be found in the openUTM manual „Programming Applications with KDCS”.

Loss of connection

The connection between two applications may be lost if errors occur on the communication link, or if one of the two applications is terminated. Loss of the connection between the partners causes the transaction to be rolled back.

When communicating via the OSI TP protocol, the job-receiving service is terminated when the connection is lost. When using the LU6.1 protocol, the job-receiving service is terminated only if it has not yet reach a synchronization point.

If the job-receiving service is terminated, the job-submitting service receives an error message.

If the job-submitting service has not yet reached a synchronization point, it too is terminated. If this occurs, it obviously does not receive any further error messages. However, openUTM sends a message to the terminal or the client program that started the jobsubmitting service.

Restarting interrupted services

If the global transaction is rolled back (e.g. due to a loss of connection, timeout, or termination of the application) without terminating the job-submitting service, a service restart is performed. In this case, communication between the client and the job-submitting service and (when using LU6.1) between the job-submitting service and the job-receiving service is resumed from the last synchronization point.

The restart takes place immediately if the client is still connected to the job-submitting application. If the client is no longer connected to the job-submitting application after the rollback procedure, e.g. because the connection to the client was shut down, the restart takes place as soon as the client signs back on to the application.

Restarting a session (only for communication via LU6.1)

A “session” is the communication relationship between applications based on the LU6.1 protocol. Each transport connection between the applications is assigned to a session. If communication is interrupted, the session concept allows you to save information on the last message sent and logged. Communication can thus be resumed at this point.

After a connection is lost, openUTM automatically attempts to restart communication at a synchronization point. A transport connection must be available for this purpose.

A session can be restarted by:

- the automatic resumption of communication when the job-submitting and the jobreceiving services are restarted
- the restart of an application, if automatic connection setup is generated for the remote application
- an administration command
- resending a message via the session

A session restart may be rejected by the remote partner if the partner is not in a position to process the restart at that time. In this case, the partner can reinitiate the session restart at a later point in time.

13.6.2 Roll-back and restart functions with independent transactions

openUTM uses local, independent transactions if the “Commit” functional unit is not used when communicating via OSI TP.

The roll-back and restart functions distinguish between the following scenarios:

- Error in the job-receiving application or loss of connection

  openUTM rolls back the transaction in the job-receiving service and terminates the service. The transaction in the job-submitting service is not automatically rolled back. However, the job-submitting service does receive an error message if it is waiting for a result from the job-receiving service, and can thus respond with a roll-back.
- Error in the job-submitting application

  openUTM rolls back the transactions in the job-submitting and job-receiving services, and terminates the job-receiving service. If the job-submitting service has already reached a synchronization point, the service is restarted after the transaction is rolled back.

13.7 Error handling for asynchronous messages

When delivering asynchronous messages, the following error situations can occur:

- Loss of connection with LTERM, LPAP, and OSI-LPAP bundles

  If a slave LTERM, slave LPAP, or slave OSI-LPAP loses its connection to the partner application, openUTM can send this slave’s pending asynchronous messages using a different slave in the bundle with an established connection, after a defined waiting time has elapsed.

  Detailed information can be found in the openUTM manual “Generating Applications” for generation parameter MAX MOVE-BUNDLE-MSGS.

- Permanent errors when sending asynchronous messages to LPAP or OSI-LPAP partners

  To prevent message loss when permanent errors occur, asynchronous messages to LPAP or OSI-LPAP can be saved in the dead letter queue (see "Control options for message queues").

  If the errors can be resolved, the messages can then be moved from the dead letter queue to the original destination or a different destination of the same type.

- Corrupt messages to asynchronous TACs and TAC queues

  Messages to asynchronous TACs and TAC queues that could not be processed can be saved in the dead letter queue to prevent message loss (see "Control options for message queues").

  If the errors can be resolved, the messages can then be moved from the dead letter queue to the original destination or a different destination of the same type.

14 openUTM on BS2000 systems

This chapter provides platform-specific details that relate specifically to the implementation of openUTM on BS2000 systems:

- system integration
- UTM processes
- address space concept
- formatting
- BS2000-specific functions

14.1 System integration

The architecture of openUTM on BS2000 systems is suited to the architecture of BS2000 systems and the openNetworking communication system.

openUTM can run without restrictions on systems with /390 architecture and on systems with x86 architecture.

UTM system code

The system code of a UTM application (the monitor) runs as part of the operating system primarily in privileged status (TPR) and uses central BS2000 and openNetworking functions.

The system code is protected by the operating system and is therefore secured against overwriting by program units.

The UTM system code is implemented as a separate subsystem (UTM) of the operating system. The system administrator can use DSSM to load this subsystem in the system memory.

The UTM system code calls the operating system functions via an adaptation module. Each version of openUTM includes adaptation modules for several BS2000 operating system versions: the correct module is loaded when the subsystem is started. This solution enables any version of openUTM to run on several operating system versions of BS2000. This also allows you to migrate your UTM applications to a later version of BS2000 operating system.

If a number of UTM applications exist in a BS2000 system, then the UTM system code is only loaded once in the BS2000 system. The entire UTM system code, including the UTM-D modules, is loaded as a subsystem of the BS2000 operating system. The system administrator uses statements for the UTM subsystem to define the time at which the UTM system code is to be loaded. If further UTM applications are started in the BS2000 system, these also use the system code already loaded. However, each UTM application is managed by openUTM using separate application tables, so that the UTM applications run fully independently of each other.

Subsystems and system functions used by the UTM system code

The UTM system code uses a range of internal system interfaces and subsystems:

- functions for requesting and releasing internal system storage areas (memory management)
- functions for job management and serialization (bourses)
- functions for managing the resources of an application (name manager)
- timer functions for monitoring resource utilization and time-driven messages
- functions of BS2000 subsystem management (DSSM) for loading and unloading the UTM system code and to start the UTM-SM2 subsystem when necessary
- data management system with the access methods UPAM and SAM
- BCAM for communicating with applications and communication partners
- VTSU for editing the messages to and from terminals
- SOC-TP when openUTM is to communicate with socket partners
- RSO (remote SPOOL output) for supporting RSO printers (see "BS2000-specific functions")
- OSS and CMX for distributed processing via OSI TP
- SAT for logging security-related UTM events

Running several versions of openUTM in parallel

It is possible to load several versions of openUTM in parallel and use them concurrently on the same BS2000 system.

This function offers considerable advantages, particularly when you are migrating to a new openUTM version. While an existing version is still being used, you can try out individual UTM applications on the successor version of openUTM. This means that you can perform step-by-step testing and migration of a number of UTM applications from one openUTM version to another on the same system.

Interfaces and components used by the main routine

In addition to the internal system interfaces, a UTM application in non-privileged processor state (TU) has interfaces to the format handling system FHS, to external resource managers such as data storage or database systems, and to the runtime systems of the programming languages used. The connections to these products are likewise technically separate and are implemented via neutral interfaces:

IUTMDB	Interface for coordination with external resource managers such as data storage or database systems. All resource managers that have an IUTMDB connection module are served in a uniform way via this interface (e.g. LEASY, SESAM or UDS).
XA	Interface to connect external resource managers such as Oracle. The XA interface is an X/Open standard.
IUTMFORM	As an interface to the FHS formatting system.
IUTMHLL	Uniform interface to the runtime systems of the individual programming languages.
ILCS	(Inter Language Communication Services) Interface to the language-independent runtime system CRTE (common runtime environment).

The program units access the functions via the program interfaces of openUTM, i.e. via theX/Open interfaces CPI-C and XATMI or via the KDCS interface (German standard).

UTM modules and utility programs as LLMs

The following components of openUTM are shipped as LLMs:

- UTM system modules
- UTM-ROOT code
- administration program
- all utility programs
- KDCUPD modules

This means that source corrections can be supplied if there is an error in a module. The module itself can then simply be replaced.

UTM utilities are loaded from a library when they are called.

You can find more information on UTM modules and utility programs in the openUTM manual “Generating Applications” and in the openUTM manual “Using UTM Applications on BS2000 Systems”.

Overview: Interfaces of openUTM

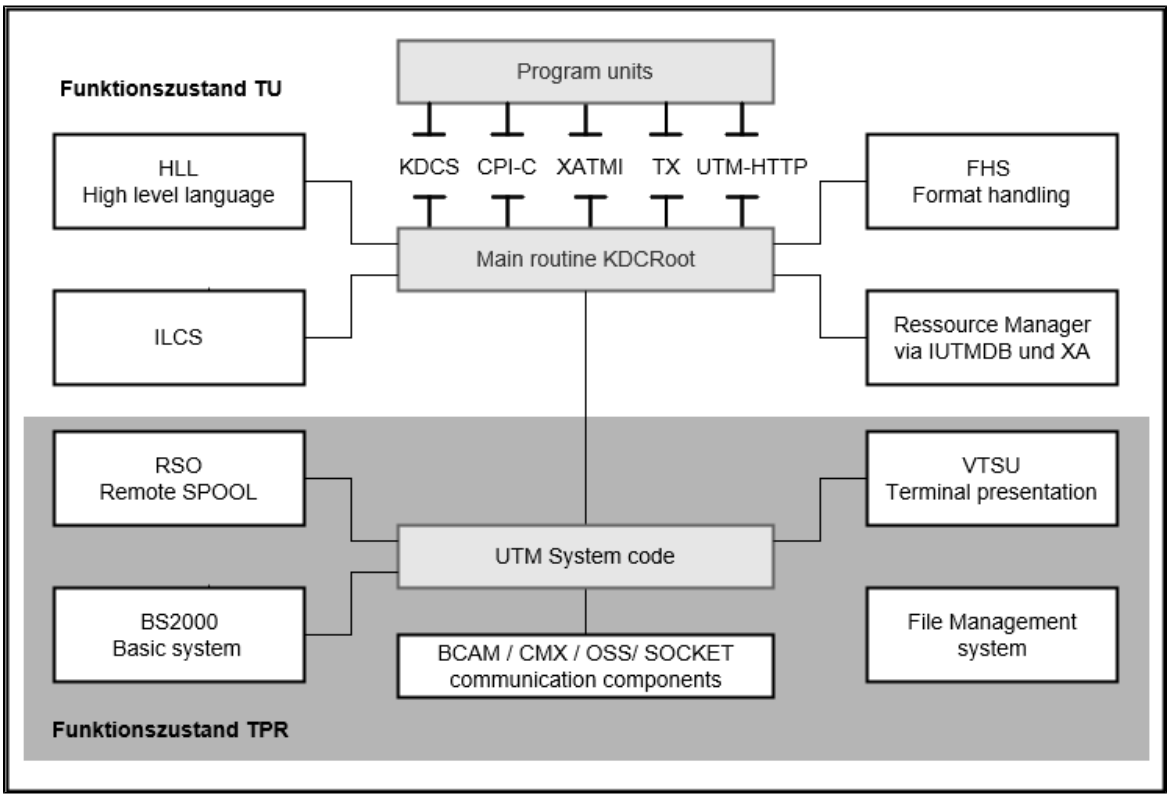


Figure 40: openUTM interfaces to other system components

14.2 UTM processes

Every UTM application running in the BS2000 system includes a homogeneous process family. In other words, all the application’s processes are identically equipped, and any process can take on any job. The processes are implemented as BS2000 tasks.

The linked application program is started as a batch job by a BS2000 procedure. The process started in this way is the first process of the UTM application and first of all configures the application. This process then activates the number of follow-up processes specified in the application generation or in the start procedure. The follow-up processes attach themselves to the existing application.

After the application start, all processes of the UTM application wait for jobs in a shared process queue. If a job arrives, it is assigned to a process waiting in the process queue. This process handles the job and then rejoins the process queue.

If there are more jobs than work processes at any one time, a job queue is established. Both the job queue and the process queue are application-specific, i.e. each application has its own process queue and job queue.

14.3 Address space concept

The storage areas used by a UTM application are stored in different BS2000 storage classes depending on the type of data.

- The UTM system code is generally located in the class 4 storage area and is therefore used jointly by all processes of all UTM applications of a BS2000 system.
- Local application system storage areas with configuration data, administrative data, and a buffer area for reducing file accesses (cache) are stored in common memory pools in the class 5 storage area. In order to relieve the load on the program address space, the cache can alternatively be created in one or more data spaces. The system storage areas of a UTM application can only be accessed by the processes of the local application and not by processes of other UTM applications.
- Each process of a UTM application is assigned a separate process-specific class 5 storage area which contains the current system data for a job or program run. This data area can only be accessed by one process and only by the UTM system functions. This data is inaccessible even to other processes within the same application, i.e. processes within the same application are isolated from each other. User programs cannot write to the area and hence cannot interfere with the execution of UTM system functions.
- The operating system assigns a user address space in the class 6 storage area to each process of a UTM application. Depending on the utilization, this user address space can be process-specific or can be shared as a common memory pool by all processes of an application (local) or by the processes of several applications (global). In general, the user address space contains:
  - The main routine KDCROOT, which establishes contact between program units and UTM system functions, as well as data and buffer areas that are shared by the UTM system functions and the main routine.
    When generating the application, parameterization of the main routine is application-specific.
  - The program units and their data areas. If program units can be used repeatedly (i.e. are shareable), then they can be stored in shared user storage areas that are local or global to the application in common memory pools or in non-privileged subsystems. This reduces the size of the external memory (paging area) occupied by the processes of the application and reduces the amount of paging, thereby enhancing performance.
  - The formatting routines for editing screen forms and printer forms and the format definitions belonging to the application. If the format handling system FHS is used, these can also be stored as sharable units in common memory pools.
  - File systems (e.g. LEASY) with associated connection module(s) for database systems such as UDS/SQL and SESAM/SQL.
  - Runtime systems of the programming languages in which the program units are encoded.
  - Administration program units for the UTM application.

The following diagram provides an overview of the storage structure of a UTM application in a BS2000 system.

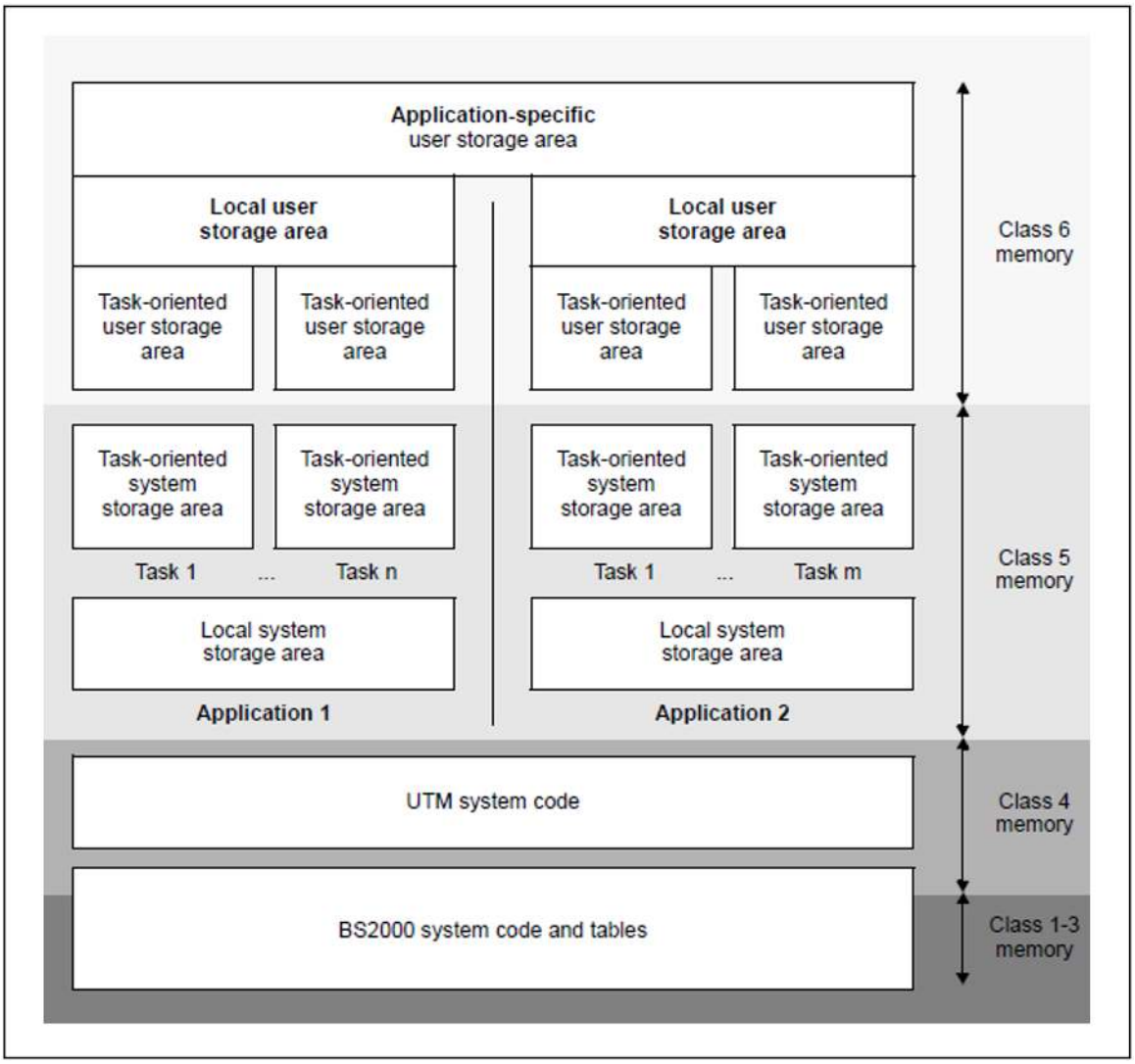


Figure 41: Storage structure of UTM applications on BS2000 systems

14.4 Formatting

If you want to operate terminals in format mode for your UTM application, you can either use the BS2000 software products IFG (interactive format generator) and FHS (format handling system), or you can create your own formatting routine (FORMAT event exit).

openUTM on BS2000 systems distinguishes between *formats, +formats, #formats, and -formats, each with different functions and implementation options. When messages are sent and received, their format is specified by means of a format identifier. This identifier consists of the format type (first character) and the format name. To format *formats, +formats and #formats, openUTM uses the FHS format handling system, and to format -formats it uses the FORMAT event exit.

IFG format generator

The interactive format generator IFG enables formats to be created quickly and easily in dialog mode. IFG automatically generates appropriate data structures (= addressing tools), which you can integrate in your program units. The data structures can also be created for data in the Unicode format. IFG also provides support for managing and maintaining your format libraries.

The interactive format generator IFG is described in a separate manual entitled “IFG”.

FHS format handling system

The FHS format handling system supports the implementation of formats created with IFG. It offers a wide range of functions, including:

- filling of message areas with freely selectable characters
- identification of the fields selected by the terminal user
- transfer of unprotected fields
- transfer of fields modified by the terminal user
- positioning of the cursor
- automatic hardcopy
- modification of display attributes
- logging of messages to restore a corrupted format
- processing of data in 7-bit code, 8-bit code, or Unicode

For communication with the FHS formatting system, openUTM uses the standard interface IUTMFORM.

IUTMFORM offers the following advantages:

- This interface separates the products openUTM and FHS, which means for example that you can use the functions of a new FHS version while retaining the same openUTM version, or vice versa.
- No FHS macros need be compiled during generation.
- UTM program units can also communicate directly with FHS. The CALL KDCFHS call is used for this purpose.
- The IUTMFORM interface is designed so that in principle it permits the connection of any formatting system and is thereby open to future developments. At present, only the FHS format handling system is supported on BS2000 systems.

You can use WebTransactions to convert FHS formats to HTML and thus integrate them in Web interfaces.

The FHS format handling system is described in a separate manual entitled “FHS - Format Handling System for UTM, TIAM, DCAM”.

FORMAT event exit

The FORMAT event exit is a formatting routine created by programmers themselves. Like the FHS formatting system standardly used by openUTM, this routine must be able to process physical input messages as well as create physical output messages - even after a screen restart. A local formatting routine is useful in the following cases:

- if you require functions above and beyond the FHS functionality supported by openUTM
- if you are using a formatting system other than FHS
- if you want to operate terminals on the physical level

The FORMAT event exit is described in the openUTM manual „Programming Applications with KDCS”.

14.5 Code conversion

Socket USP applications and Http clients normally use ASCII compatible code, while the EBCDIC code is used on BS2000 systems. openUTM therefore offers an automatic code conversion on BS2000 systems for socket USP partners and HTTP clients.

openUTM provides four code conversions, two 8-bit conversions and two 7-bit conversions. You can modify these conversion tables.

You will find more information on code conversion in the openUTM manual “Generating Applications” under the PTERM, TPOOL and CHAR-SET statements and in the appendix.

14.6 BS2000-specific functions

Local and central connection of printers

Printers can be connected in two different ways:

- “centrally” as a network printer
- “locally” to a terminal

Network printers must be generated explicitly in UTM and BCAM. In the case of locally connected printers, only the terminal need be taken into account in the generation. The connection type has no effect on the program interface.

Regardless of the connection type, openUTM supports the following operating modes:

- hardcopy mode
- spool mode

If a printer is connected locally, spool mode is referred to as bypass mode. In bypass mode, the terminal can manage a dialog independently of the print output. Bypass mode can only be implemented for particular terminal types (e.g. 9763 terminals).

Using RSO printers

UTM applications on BS2000 systems can also use RSO printers. The OLTP interface made available by RSO (Remote Spool Output) gives openUTM access to all printers supported by RSO, i.e. even to printers connected via LAN or to PCs. openUTM does not establish a transport connection to these printers; instead, it accesses them via the OLTP interface, i.e. openUTM reserves the printer in the RSO and then passes the print job to the RSO.

At the program interfaces, RSO printers are handled in exactly the same way as printers that are connected in a different manner. Print options can also be passed to the printer in the form of a parameter list.

There is a separate series of manuals for the RSO software product provided by BS2000 systems. Further information specific to UTM generation can be found in the openUTM manual “Generating Applications” under the heading “RSO”.

Sharing printers

You can use a generation parameter (PLEV parameter of the KDCDEF statement LTERM) to allow a printer to be used by a number of UTM applications. This is achieved by maintaining the connection between a UTM application and the printer for short periods only, thereby giving other UTM applications the option of setting up a connection. If this parameter is used, openUTM only sets up a connection to a printer when the number of messages waiting to be printed exceeds the threshold value generated for a specific printer. The connection is shut down again when no more messages are waiting to be printed.

Run priorities

During generation you can assign each transaction code an individual run priority withon BS2000 systems. This run priority is assigned to the UTM process in which the program unit is running. As a result, you can use BS2000 system’s scheduling mechanisms to control the execution of UTM program units.

Specific security functions

SAT logging

You can log security-related UTM events with the BS2000 function SAT (security audit trail). This log provides the verification required in accordance with the F2/Q3 criteria of the ITS catalog.

Additional system access control through connection passwords

Each UTM application in a BS2000 system can be protected against unauthorized use by a connection password that can be defined when the application is started. Each terminal user who wants to work with this UTM application must specify this password.

Support for Kerberos and system access control using Kerberos

On BS2000 systems, when establishing a connection from terminals, a Kerberos dialog can be executed, the results of which can be read in the program unit. This functionality can be generated using the LTERM and TPOOL statements.

System access control using the Kerberos distributed authorization service can be generated using the USER statement.

Database keys

For UTM applications on BS2000 systems, a database key can be assigned to a transaction code in the generation if they use the IUTMDB interface, see "System integration". This means that certain access rights to the database can be assigned to the TAC. openUTM transfers the key to the database system, where it is analyzed by certain database systems in order to check the access rights to database records.

Encryption for terminal connection

Encryption can also be used for terminal emulations in UTM applications on BS2000 systems. The encryption is executed on the host side by the BS2000 product VTSU-B. In this fashion, every emulation can be operated with encryption as long as it supports the corresponding functions.

Further details on the BS2000-specific security functions outlined above can be found in the openUTM manual “Generating Applications”.

Internationalization / XHCS support

openUTM supports internationalization:

A UTM application can be generated such that communication partners using different languages can be served in their particular language. Even regional differences within a language can be taken into account. Dates, times, units of measurement, and currency symbols can be displayed in accordance with local conventions.

On BS2000 systems you can assign a particular language environment – also known as the “locale” – to the individual user IDs, access points (LTERM partners), or the entire application when configuring a UTM application. This locale is specified as follows:

LOCALE=(*language-code*, *territorial-code*, *character-set-name*)

The program units of the UTM application can access the locale information and interpret the input of the communication partner or create messages to the communication partner as appropriate.

In addition, on BS2000 systems you can generate a number of message modules for a UTM application and thereby also have multilingual UTM messages.

openUTM on BS2000 systems thus provides a full range of internationalization options. A similar function is available on Unix and Linux systems by using NLS (native language support).

To display the fonts and special characters of the individual languages on terminals or printers, various character sets (8-bit codes or unicode character sets such as UTFE) may be required. A number of character sets can be used simultaneously in a BS2000 system with the aid of the BS2000 software product XHCS (eXtended Host Code Support).

openUTM supports the functions of XHCS. This means that a specific extended character set, which is then used for formatting messages, can be assigned to individual user IDs, individual access points (LTERM partners), and the entire application. In particular, Unicode data can be processed in the application program, and FHS formats with Unicode data can be read in and output.

Further information on XHCS can be found in the User Guide “XHCS V2.0 - 8-Bit Code and Unicode Support on BS2000/OSD”. The UTM-specific aspects of internationalization are described in the openUTM manual “Generating Applications”.
For details on implementing a number of message modules, see the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

Using the OMNIS session manager

The services of the BS2000 software product OMNIS can be used for UTM applications on BS2000 systems. OMNIS is a session manager that enables a terminal user to call the services of various UTM applications directly, even if the UTM applications are distributed throughout the network. In this case, the terminal user does not have to know the computer or UTM application in which the service is running: OMNIS automatically establishes a connection to the “correct” UTM application and regulates the assignment of messages (message distribution).

If you are using OMNIS, you can also use the multiplex function provided by openUTM on BS2000 systems: a large number of terminals can link up with a UTM application via a small number of transport connections.

The add-on product OMNIS-MENU is available if you want to use OMNIS via menus.

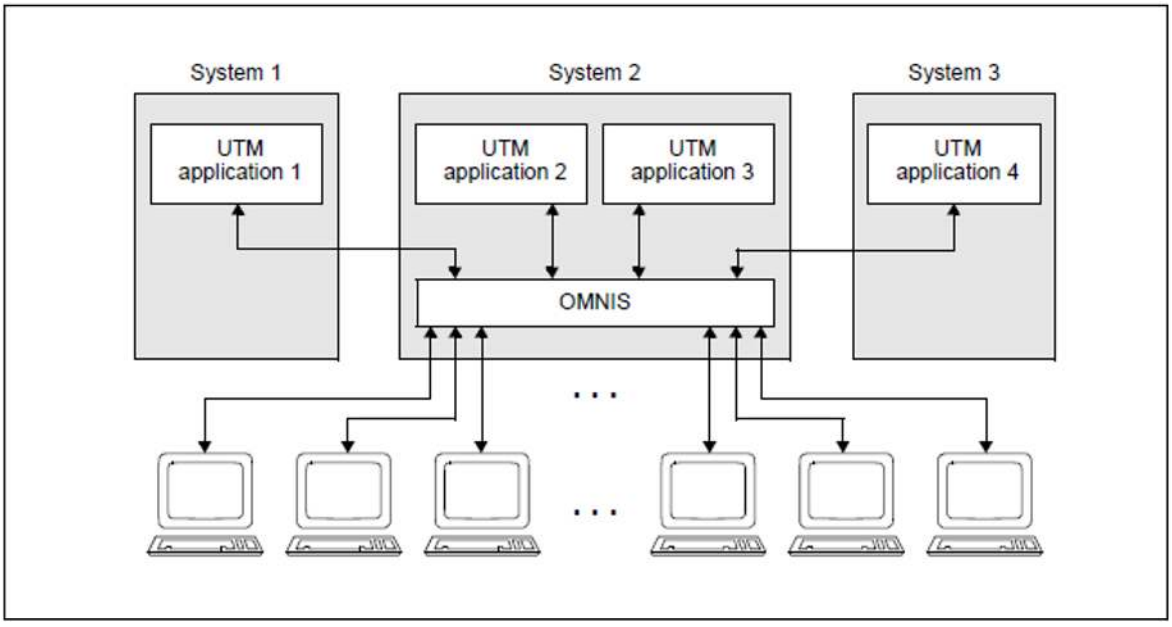


Figure 42: Message distribution and multiplexing with OMNIS

Separate manuals are available on the OMNIS session manager and OMNIS menu control: “OMNIS /OMNIS-MENU Functions and Commands” and “OMNIS/OMNIS-MENU Administration and Programming”.

For information on defining multiplex connections when generating UTM (see the openUTM manual “Generating Applications”).

Calling UTM services with CALLUTM

The program CALLUTM is supplied with openUTM on BS2000 systems. CALLUTM allows UTM services to be called from within any BS2000 batch task or dialog task. The program has an SDF interface and can itself be called from within procedures. One of the uses to which the program can be put is to call UTM administration commands, e.g. to terminate all UTM applications with KDCSHUT subject to procedure control. In this way, you can administer one or more UTM applications irrespective of the computer or operating system on which they are running.

CALLUTM is a UPIC client in the BS2000 system and communicates with the UTM applications via the CPI-C interface with UPIC as carrier system. CALLUTM can therefore utilize the UTM user concept, i.e. it can sign on using a UTM user ID that can also be a Password-protected.

For more information on CALLUTM, see the openUTM manual “Administering Applications”.

SDF interface for UTM tools

UTM tools such as KDCDEF can be started using separate SDF commands. The commands are located in the SDF UTM application area.

For a description of these SDF commands, see openUTM manual “Using UTM Applications on BS2000 Systems”.

15 openUTM on Unix and Linux systems

This chapter provides platform-specific details that relate specifically to the implementation of openUTM on Unix and Linux systems:

- system integration
- UTM processes
- address space concept
- configuration of the network connection

15.1 System integration

To achieve the greatest possible level of portability, the UTM system code only uses the system calls and library functions provided by Unix and Linux systems within the framework of the X/Open universe (system V):

- functions for requesting and releasing internal system storage areas (local process storage area and shared memories)
- functions for serialization (semaphores)
- functions for creating and terminating processes
- functions for timing resources and time-driven messages
- functions for editing files and databases

In addition to the interfaces to the operating system, a UTM application has a range of other internal interfaces:

- XA interface (X/Open standard) for connecting external resource managers(such as Oracle, INFORMIX)
- UPIC-L interface which enables UTM client programs to be connected locally to the UPIC carrier system (i.e. the client programs can run in the same Unix or Linux system as the UTM application)
- interfaces to the runtime systems of the programming languages used
- interfaces to the communication components PCMX

The program units access the functions via the program interfaces of openUTM, i.e. via theX/Open interfaces CPI-C and XATMI + TX or via the KDCS interface (German standard).

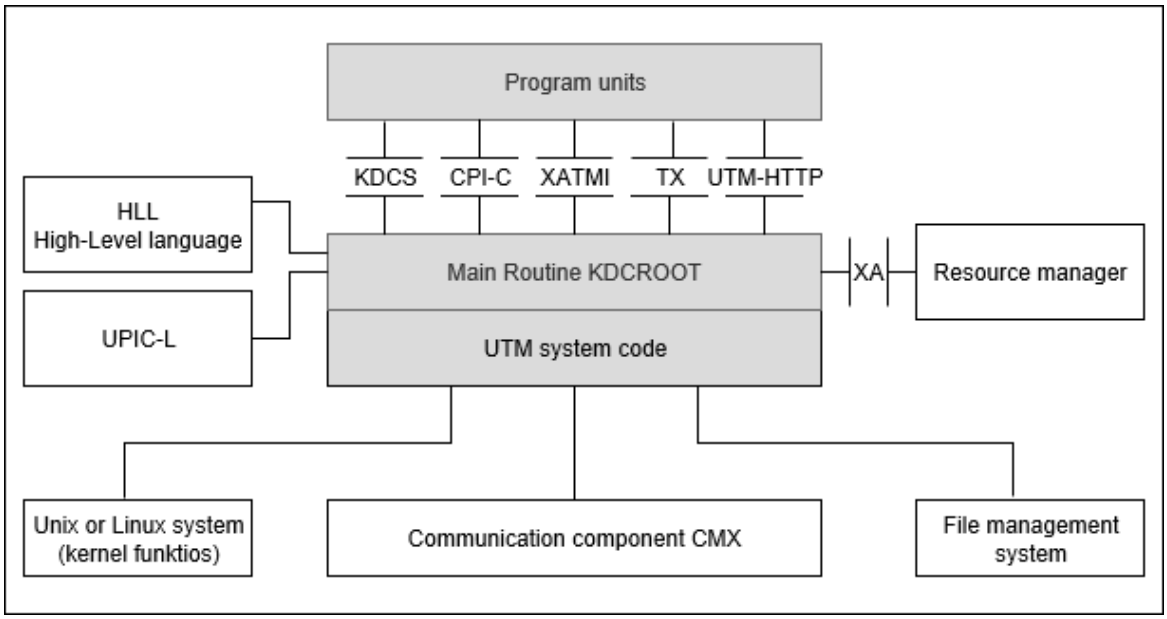


Figure 43: openUTM interfaces to other system components

openUTM is executable on 32-bit and 64-bit platforms, but a mixed operation of 32-bit and 64-bit is not possible within a UTM application.
When the application is started and a utility is started, the system checks whether the components used are compatible with each other with regard to the platform and the bit mode.

15.2 UTM processes

Different processes work together when a UTM application program running on a Unix or Linux system is executed, whereby each process performs specific tasks. These process types are described below. An overview is provided in figure 44.

Main process and work processes

A UTM application is started by the *utmmain* program. This program is generally started as a background process known as the **main process**. The main process then creates as many **work processes** as are specified in the start parameters. The application program created by the user is loaded and started in all of these work processes.

The work processes perform the actual work, i.e. they handle the service requests sent to the UTM application. The main process monitors these productive processes. During the application run, the main process automatically creates further work processes if a work process is terminated due to an error or if additional work processes are explicitly allocated to the application by administration.

After the application start, all the work processes of the UTM application wait for jobs in a shared process queue. If a job arrives, it is assigned to a work process waiting in the queue. This process handles the job and then rejoins the process queue.

If more jobs exist than work processes, a job queue is established. Both the job queue and process queue are application-related, i.e. each application has its own process queue and job queue. The queues for processes are implemented by semaphores, while the queues for jobs are implemented by a shared memory.

Timer process

In addition to the work processes, the main process sets up a **timer process** which is assigned to the application. The timer process accepts jobs from the work processes in order to time the wait states, and arranges these jobs in a job queue. After one of the times recorded in the job queue expires, this is indicated to the work processes for processing.

Network processes

With distributed processing, UTM applications are connected to the network via **network processes**. The task of these processes is to process connection requests and manage the data transfer on this connection.

The network connection can run via PCMX or directly through the socket interface. The number of network processes depends on the generation.

For further information on network processes and for details on generation, see the openUTM manual “Generating Applications”.

Dialog terminal processes (DTPs)

Each terminal which works with the UTM application has its own dialog process, known as the **dialog terminal process**. This is created neither by the main process nor a work process, rather is established from the shell by starting the *utmdtp* program or is started automatically when the user successfully logs on to the Unix or Linux system.

The terminal user selects the UTM application, thereby establishing a connection between the dialog terminal process and the UTM application. The dialog terminal process can then send jobs to the UTM application and receive messages from the UTM application.

Local client processes

A separate local client process exists for each UTM client that is based on the UPIC carrier system and works with the UTM application. These processes are created neither by the main process nor by the work process but from the shell.

The local client process establishes a connection to the UTM application. The local client process can then send jobs to the UTM application and receive messages from the UTM application.

Printer processes

Asynchronous messages to printers are output by the UTM application using local processes known as **printer processes**. The main process of the UTM application sets up a printer process for each connected printer. The printer process for a printer exists for as long as this printer is connected to the UTM application.

Logging process

openUTM can record certain data such as accounting records or event data while the application is running. The recording of data is controlled by the **logging process** (*utmlog*).

- When generated, openUTM provides accounting information during the application run. This information is recorded in "accounting records" by openUTM and forwarded to the logging process. The logging process writes these records in a file in the ACCNT subdirectory.
- The UTM event monitor KDCMON functionality integrated into openUTM is available to monitor the performance of an openUTM application. After starting the KDCMON monitor, the work processes record the event data and pass it to the logging process, which then writes the data records in a file in the KDCMON subdirectory. The task of administering this data is then moved from the work processes to a single instance, the logging process.

Overview: Processes of a UTM application on Unix and Linux systems

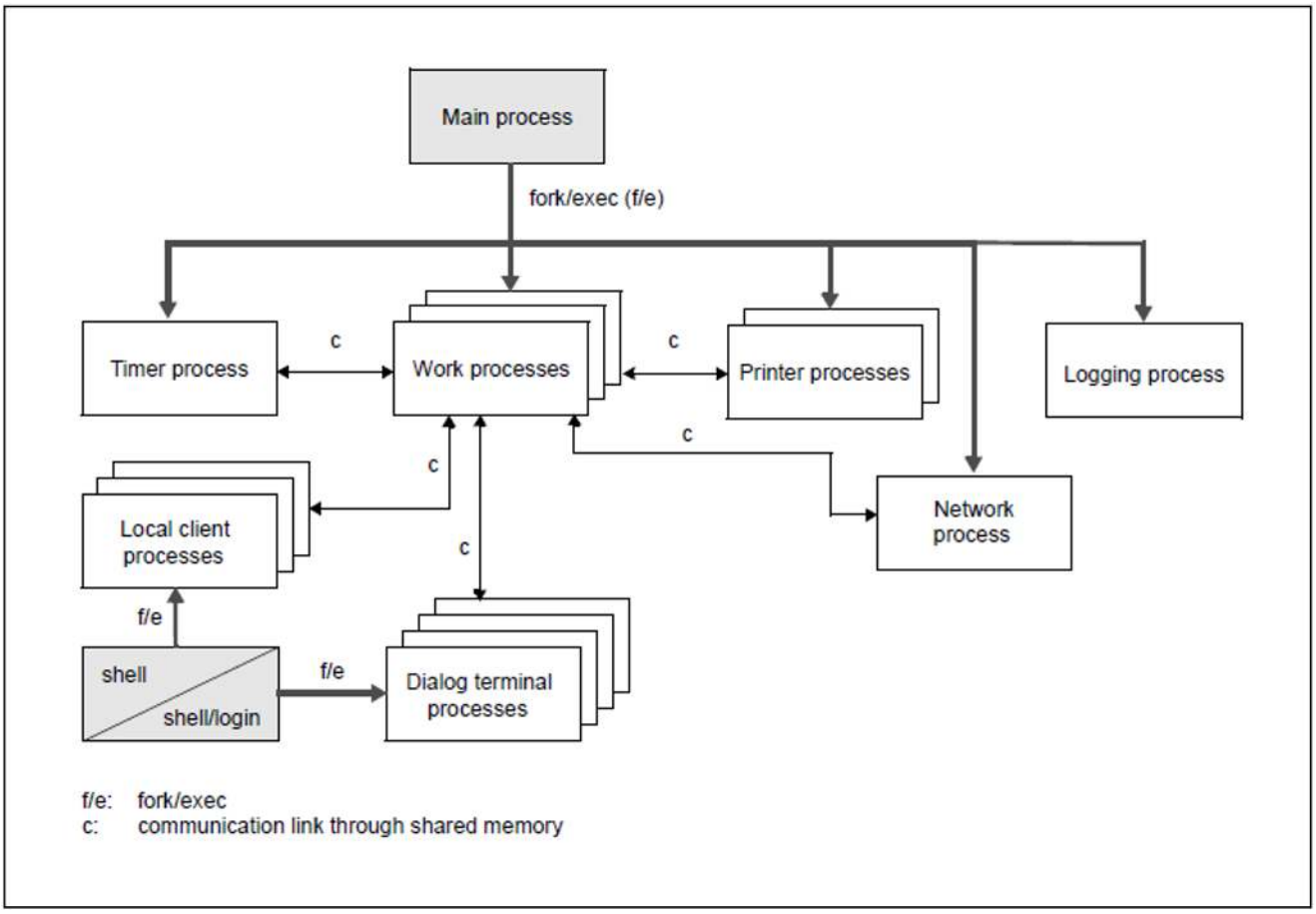


Figure 44: Process interaction in a UTM application on Unix and Linux systems

15.3 Address space concept

In a UTM application, each work process has a process-specific storage area containing:

- the data area ROOTDATA for communication between KDCROOT and the system functions
- the KB and SPAB areas
- buffer areas for MPUT messages
- a trace area for KDCS calls for diagnostic purposes
- tables for activating the program units
- the data area KTA (KDCS task area), which is only used by the UTM system functions. This contains further buffer areas, an internal UTM trace area, and various processor-specific check data

All work processes of a UTM application use a shared memory which contains the configuration and global administrative data (KAA = KDCS application area), as well as a shared memory for a transaction-oriented cache area for optimizing file access.

Work processes and external processes (dialog terminal, printer, network processes and timer process as well as local client processes) use the same shared memory area for interprocess communication (IPC) and job processing.

Because the Unix or Linux system does not have any special security mechanisms for application programs, it must be remembered that errors in the program units created by the user can also destroy UTM system areas.

The following diagram illustrates the relationships between shared memories and the processes of a UTM application.

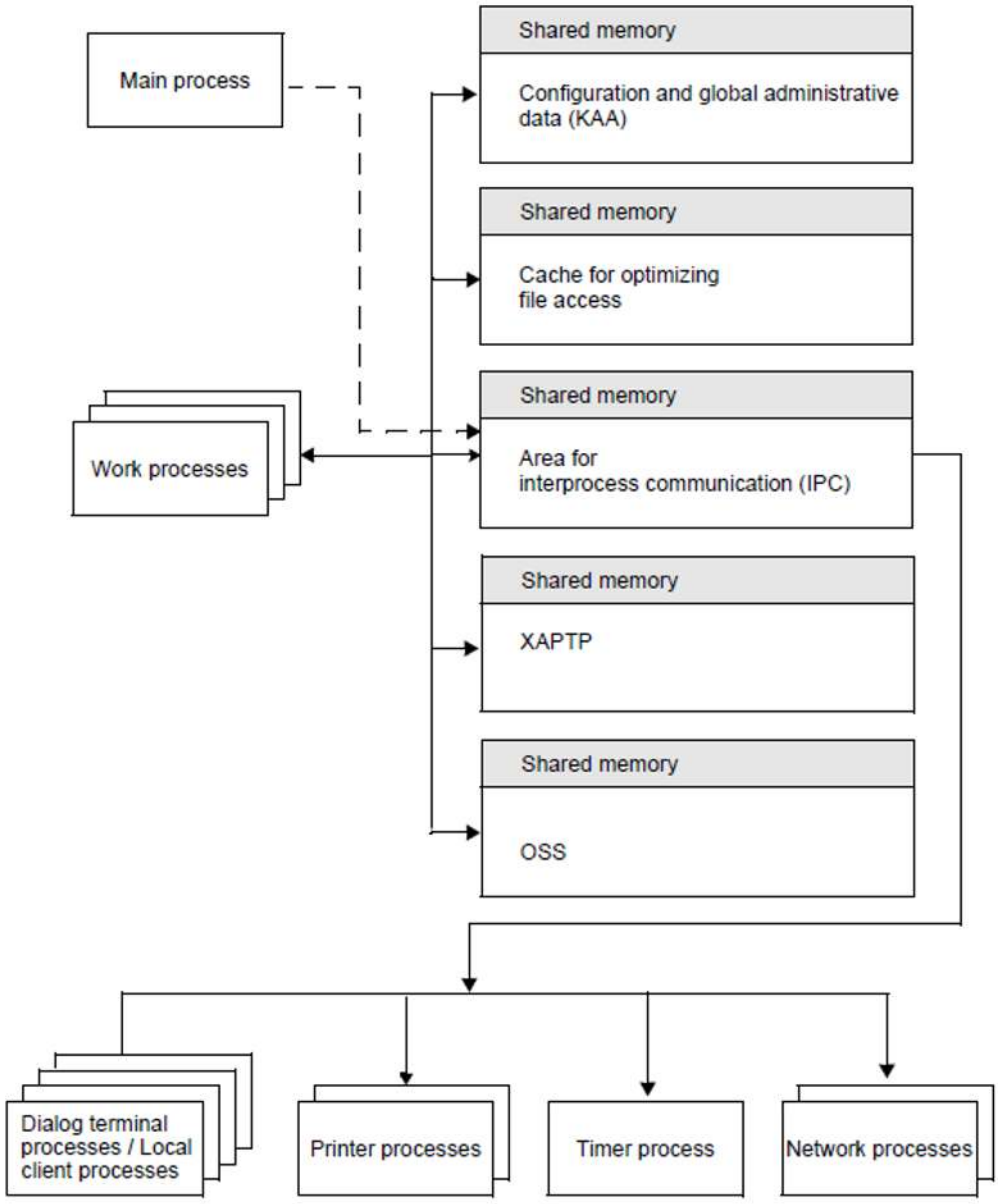


Figure 45: Shared memories and processes in UTM applications on Unix and Linux systems

15.4 Configuration of the network connection

The user enters the necessary parameters for the only two relevant couplings into the UTM generation via TCP/IP-RFC1006 and via socket. The parameters are then taken from the UTM generation at run time; the IP addresses are determined when the application is started from the corresponding host file or host database. In addition, whenever a connection is established, the currently valid addresses are determined and are used from that point onwards.

In order for a UTM application to be operated without interruption when an address is changed in the network, openUTM provides a corresponding administration function. With this function IP addresses can be read from the host database and can be transmitted to the UTM application at run time.

The openUTM manual “Generating Applications” contains detailed descriptions of how to configure network connections.

15.5 Code conversion

When messages of a UTM application are exchanged with a partner application, openUTM can carry out ASCII-EBCDIC code conversion automatically. openUTM uses a standard table for the conversion. You can modify this standard table.

You will find more information on code conversion in the openUTM manual “Generating Applications” under the statements PTERM and TPOOL and in the appendix.

15.6 User-specific error handling

The user can program his own error handling, which is called instead of the standard error handling provided by openUTM. This optimizes the high availability of the application and improves fault diagnosis.

in detail, the following is possible without terminating the current process:

- Call of a separate error routine, which e.g. outputs a message to the communication partner.
- Creation of diagnostic documents in the form of a UTM dump.
- Reload the application program after the normal end of the transaction.
-

You will find more information on user specific error handling in the openUTM manual „Programming applications with KDCS“.

16 openUTM on Windows systems

This chapter provides platform-specific details that relate specifically to the implementation of openUTM on Windows systems:

- system integration
- UTM processes
- address space concept
- configuration of the network connection

Restrictions

Formatting and transaction-oriented printer output is not supported on Windows systems.

16.1 System integration

In addition to the interfaces to the Windows operating system, a UTM application has a range of other internal interfaces:

- XA interface (X/Open standard) for connecting external resource managers(such as Oracle,...)
- UPIC-L interface which enables UTM client programs to be connected locally to the UPIC carrier system (i.e. the client programs can run in the same Windows system as the UTM application)
- interfaces to the runtime systems of the programming languages used
- interfaces to the communication component PCMX

Furthermore, **Windows Event Logging** is supported during operation with openUTM. This means that the installation, deinstallation and operation of openUTM services are logged as events in the Event Viewer. openUTM events can be seen in the Application section of the Event Viewer. The source is openUTM.

The program units access the functions via the program interfaces of openUTM, i.e. via theX/Open interfaces CPI-C and XATMI + TX or via the KDCS interface (German standard).

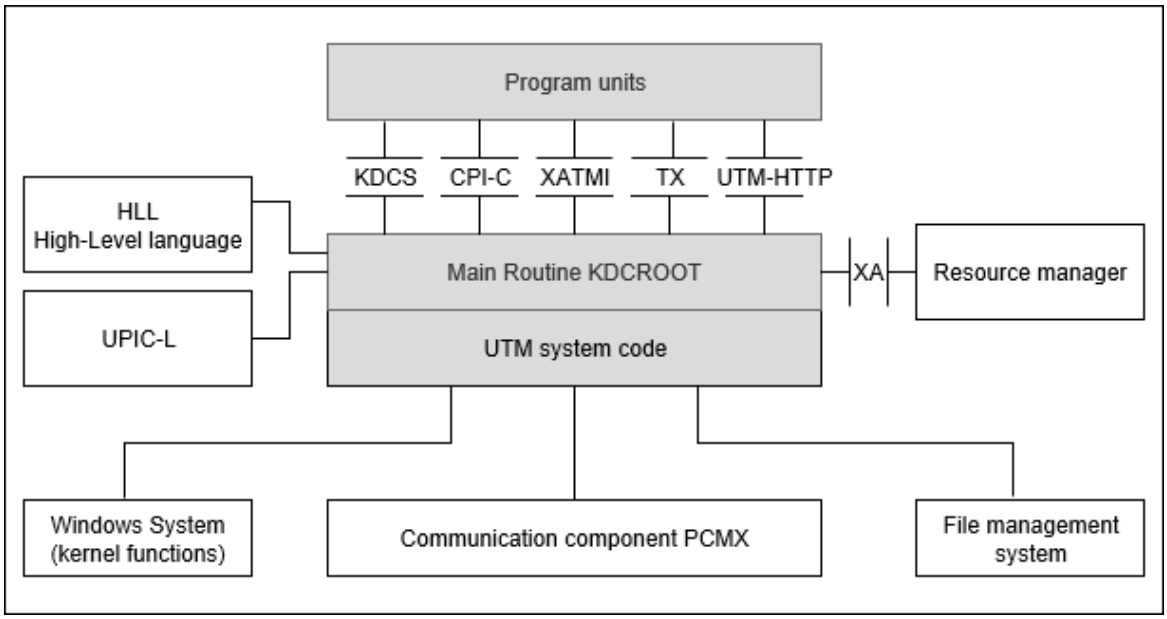


Figure 46: openUTM interfaces to other system components

openUTM is executable on 32-bit and 64-bit platforms, but a mixed operation of 32-bit and 64-bit is not possible within a UTM application.
When the application is started and a utility is started, the system checks whether the components used are compatible with each other with regard to the platform and the bit mode.

16.2 UTM processes

A UTM application is created on Windows systems as a Windows console application. Different processes work together when a UTM application program is executed, whereby each process performs specific tasks. Some of these processes are started in a DOS window (prompt) by calling the program. Shortcuts can be created for these program calls so that the processes can be started using the mouse or by entering commands through the keyboard.

The various process types are described in the following paragraphs. figure 47 shows an overview of these process types.

Main process and service process

A UTM application is started by setting up the **main process**. The main process can run in the background or in the foreground.

The main process is started in the foreground by calling the *utmmain* program. *utmmain* is called from a CMD window. This type of application start can especially be used during the development of the application.

A UTM application can be started using the *utmmains* program as a service for production operation.This process is therefore called the **service process**. It starts the main process, which then runs in the background. All output is redirected to a file in this case. If the application is set up as a service, then it can also be started automatically after the Windows system is started.

Work processes

The main process starts as many **work processes** as are specified in the start parameters. The application program created by the user is loaded into all of these work processes and then started.

The work processes perform the actual work, i.e. they handle the service requests sent to the UTM application. The main process monitors these productive processes. During the application run, the main process automatically creates further work processes if a work process is terminated due to an error or if additional work processes are explicitly allocated to the application by administration.

After the application start, all the work processes of the UTM application wait for jobs in a shared process queue. If a job arrives, it is assigned to a work process waiting in the queue. This process handles the job and then rejoins the process queue.

If more jobs exist than work processes, a job queue is established. Both the job queue and process queue are application-related, i.e. each application has its own process queue and job queue. The queues for processes are implemented by semaphores, while the queues for jobs are implemented by a shared memory.

Timer process

In addition to the work processes, the main process sets up a **timer process** which is assigned to the application. The timer process accepts jobs from the work processes in order to time the wait states, and arranges these jobs in a job queue. After one of the times recorded in the job queue expires, this is indicated to the work processes for processing.

Network processes

With distributed processing, UTM applications are connected to the network via **network processes**. These processes are set up by the main process, and the task of these processes is to process connection requests and manage the data transfer on this connection.

The network connection can run via PCMX or directly through the socket interface. The number of network processes depends on the generation.

For further information network processes and for details on generation, see the openUTM manual “Generating Applications”.

You will find a description of how to control net processes using environment variables in the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

Dialog terminal processes (DTPs)

Each console window which works with the UTM application has its own dialog process, known as the **dialog terminal process**. This is established from the DOS shell by starting the *utmdtp* program, or it is started automatically when the user successfully logs on to the Windows system by making the appropriate entry in the Startup group.

The user selects the UTM application, thereby establishing a connection between the dialog terminal process and the UTM application. The dialog terminal process can then send jobs to the UTM application and receive messages from the UTM application.

A dialog terminal process can only be started on the computer on which the UTM application is running.

Shutdown process

The shutdown process *utmshut* is set up when the application is started. This process ensures that the UTM application is properly terminated when the system shuts down.

Local client processes

A separate local client process exists for each UTM client that is based on the UPIC carrier system and works with the UTM application. This process is started for example from the Windows command prompt.

The local client process establishes a connection to the UTM application. The local client process can then send jobs to the UTM application and receive messages from the UTM application.

Logging process

openUTM can record certain data such as accounting records or event data while the application is running. The recording of data is controlled by the **logging process** (*utmlog*).

- When generated, openUTM provides accounting information during the application run. This information is recorded in "accounting records" by openUTM and forwarded to the logging process. The logging process writes these records in a file in the ACCNT subdirectory.
- The UTM event monitor KDCMON functionality integrated into openUTM is available to monitor the performance of an openUTM application. After starting the KDCMON monitor, the work processes record the event data and pass it to the logging process, which then writes the data records in a file in the KDCMON subdirectory. The task of administering this data is then moved from the work processes to a single instance, the logging process.

Overview: Processes of a UTM application on Windows systems

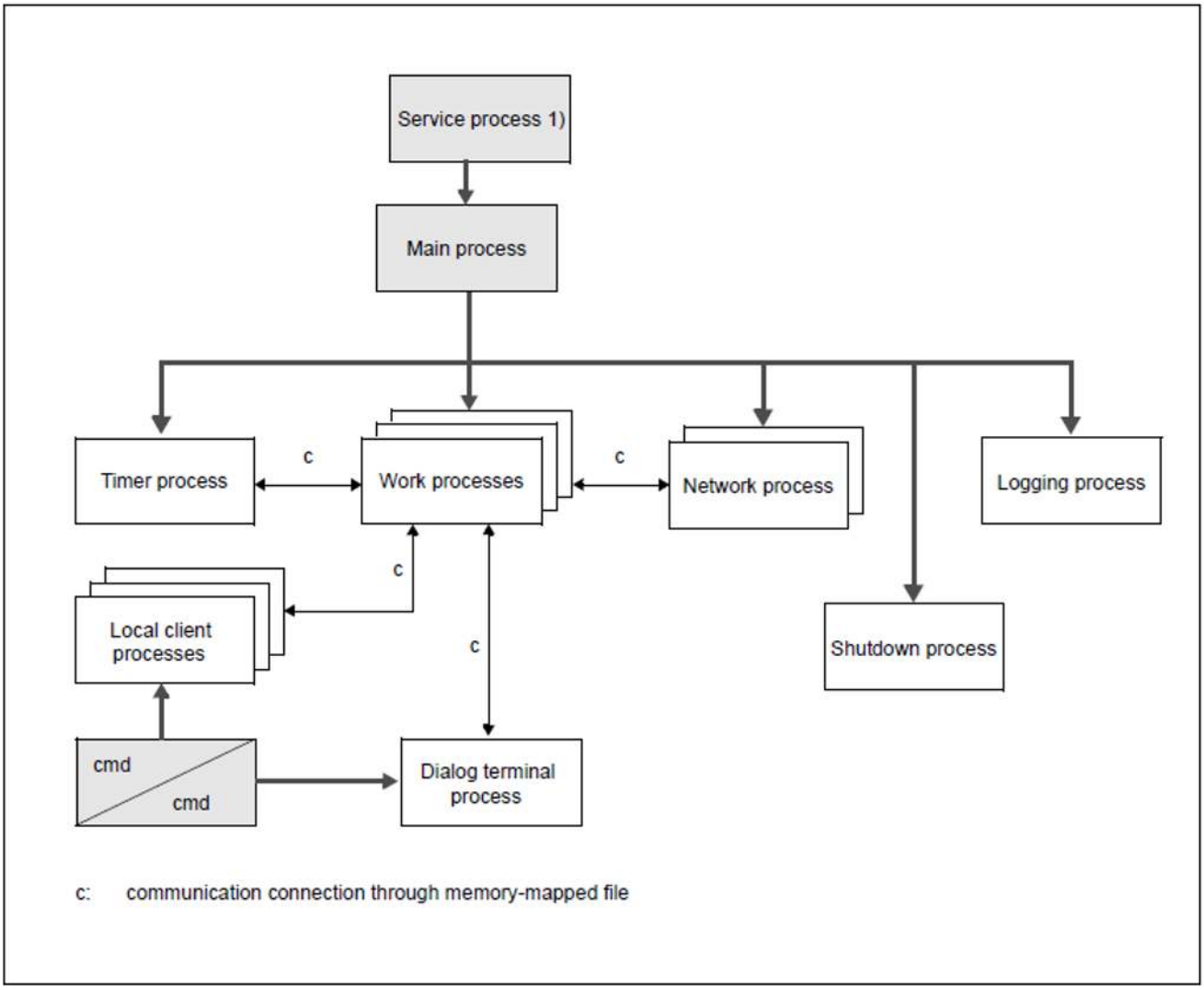


Figure 47: Process interaction in a UTM application on Windows systems

[1)]The service process is optional. It exists only if the UTM application is started as service.

16.3 Address space concept

In a UTM application, each work process has a process-specific storage area containing:

- the data area ROOTDATA for communication between KDCROOT and the system functions
- the KB and SPAB areas
- buffer areas for MPUT messages
- a trace area for KDCS calls for diagnostic purposes
- tables for activating the program units
- the data area KTA (KDCS task area), which is only used by the UTM system functions. This contains further buffer areas, an internal UTM trace area, and various processor-specific check data.

All work processes of a UTM application share a common *memory-mapped file* that contains the configuration data and global application administration data (KAA = KDCS Application Area) as well as a *memory-mapped file* for a cache area for optimizing file accesses.

Work processes and external processes (dialog terminal processes, network processes, the timer process and the local client process) share a *memory-mapped file* area for IPC (Inter-Process Communication) and for job processing.

Note that errors in the program units created by the user can also destroy UTM system areas because the Windows system does not provide a special protection mechanism for user programs.

The following figure shows the relationship mentioned between the *memory-mapped files* and the processes of a UTM application.

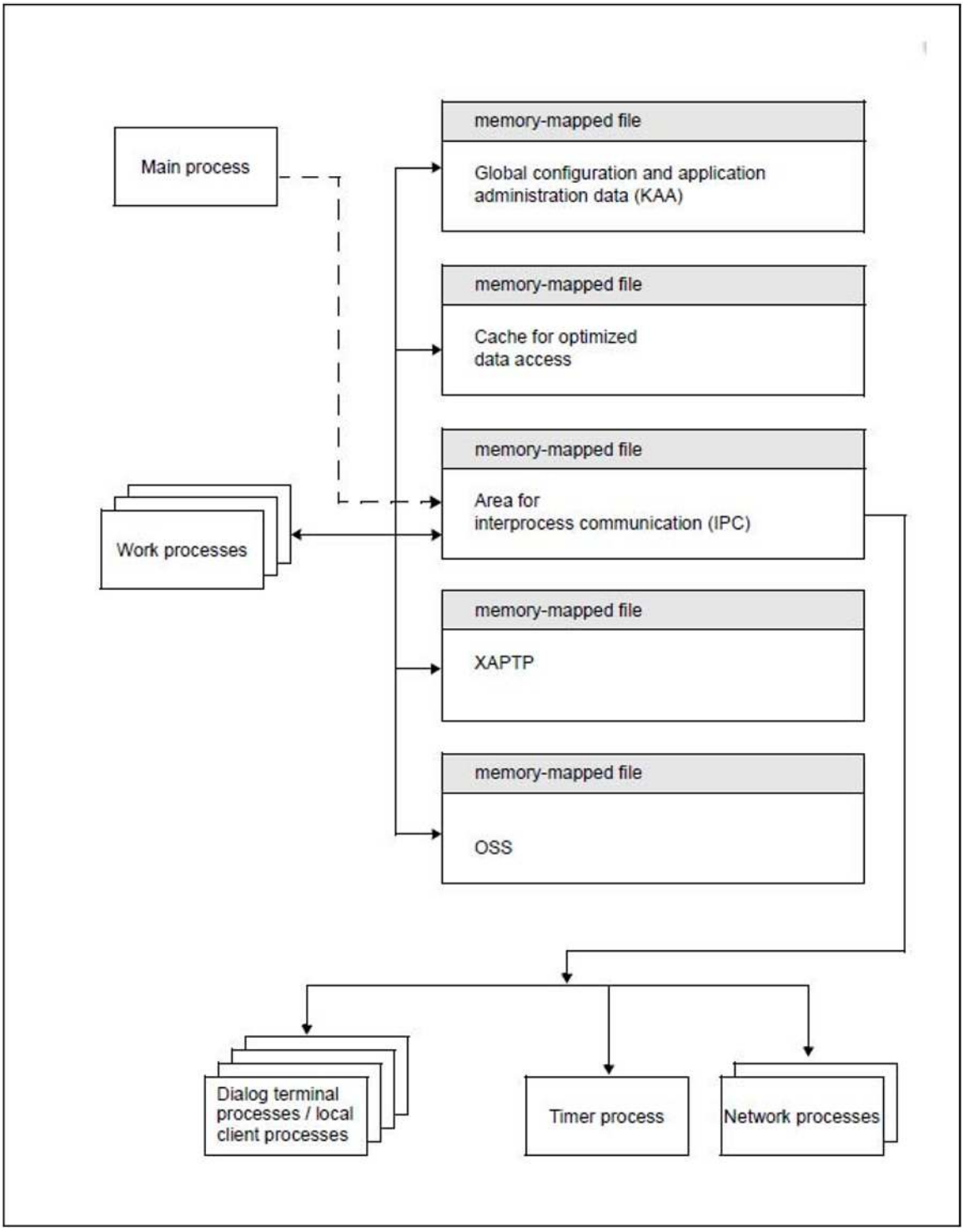


Figure 48: memory-mapped files and processes in UTM applications on Windows systems

16.4 Configuration of the network connection

The user enters the necessary parameters for the two relevant couplings into the UTM generation via TCP/IP-RFC1006 and via socket. The parameters are then taken from the UTM generation at run time; the IP addresses are determined when the application is started from the corresponding host file or host database. In addition, whenever a connection is established, the currently valid addresses are determined and are used from that point onwards.

In order for a UTM application to be operated without interruption when an address is changed in the network, openUTM provides a corresponding administration function. With this function IP addresses can be read from the host database and can be transmitted to the UTM application at run time.

The openUTM manual “Generating Applications” contains detailed descriptions of how to configure various network connections.

16.5 Code conversion

When messages of a UTM application are exchanged with a partner application, openUTM can carry out ASCII-EBCDIC code conversion automatically. openUTM uses a standard table for the conversion. You can modify this standard table.

You will find more information on code conversion in the openUTM manual “Generating Applications” under the statements PTERM and TPOOL and in the appendix.

17 Appendix: Supported standards and norms

Program interfaces

ISO/IEC 9805-1:1994 (CCR Protocol)

ISO/IEC 10026-3:1996 (OSI TP Protocol, Second Edition)

ISO/IEC ISP 12061-1:1995 (OSI TP Taxonomy)

ISO/IEC ISP 12061-2:1995 (OSI TP Support of OSI TP APDUs)

ISO/IEC ISP 12061-3:1995 (OSI TP Support of CCR APDUs)

ISO/IEC ISP 12061-4:1995 (OSI TP Support of Session, Presentation and ACSE PDUs)

ISO/IEC ISP 12061-5:1995 (OSI TP Profile ATP11)

ISO/IEC ISP 12061-7:1995 (OSI TP Profile ATP21)

ISO/IEC ISP 12061-9:1995 (OSI TP Profile ATP31)

X/Open Distributed TP: Reference Model, Version 3, G504 2/96 (X/Open Guide)

X/Open Distributed TP: The XA Specification, C193 2/92

X/Open Distributed TP: The TX (Transaction Demarcation) Specification, C504 4/95

X/Open Distributed TP: The XATMI Specification, C506 11/95

X/Open Distributed TP: The XCPI-C Specification, Version 2, C419 12/95

X/Open ACSE/Presentation: Transaction Processing API (XAP-TP), C409 4/95

RFC 7230 - HTTP 1.1 - Message Syntax and Routing, IETF, HTTP Working Group, Juni 2014

RFC 7231 - HTTP 1.1 - Semantics and Content, IETF, HTTP Working Group, Juni 2014

RFC 7232 - HTTP 1.1 - Conditional Requests, IETF, HTTP Working Group, Juni 2014

RFC 7233 - HTTP 1.1 - Range Requests, IETF, HTTP Working Group, Juni 2014

RFC 7234 - HTTP 1.1 - Caching, IETF, HTTP Working Group, Juni 2014

RFC 7235 - HTTP 1.1 - Authentication, IETF, HTTP Working Group, Juni 2014

RFC5789 - PATCH Method for HTTP, IETF, HTTP Working Group, März 2010

DIN standard 66 265: Interfaces of a kernel for transaction-oriented application systems (KDCS-TAS kernel)

XAP-TP interfaces

ISO/IEC 9805-1:1994 (CCR Protocol)

ISO/IEC 10026-3:1996 (OSI TP Protocol, Second Edition)

ISO/IEC ISP 12061-1:1995 (OSI TP Taxonomy)

ISO/IEC ISP 12061-2:1995 (OSI TP Support of OSI TP APDUs)

ISO/IEC ISP 12061-3:1995 (OSI TP Support of CCR APDUs)

ISO/IEC ISP 12061-4:1995 (OSI TP Support of Session, Presentation and ACSE PDUs)

ISO/IEC ISP 12061-5:1995 (OSI TP Profile ATP11)

ISO/IEC ISP 12061-6:1995 (OSI TP Profile ATP12)

ISO/IEC ISP 12061-7:1995 (OSI TP Profile ATP21)

ISO/IEC ISP 12061-8:1995 (OSI TP Profile ATP22)

ISO/IEC ISP 12061-9:1995 (OSI TP Profile ATP31)

ISO/IEC ISP 12061-10:1995 (OSI TP Profile ATP32)

X/Open ACSE/Presentation: Transaction Processing API (XAP-TP), C409 4/95

18 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 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 *KDCFILE* and the ROOT table sources for the main routine *KDCROOT*.
In UTM cluster applications, KDCDEF also creates the *cluster configuration file,* the *cluster user file*, the *cluster page pool*, the *cluster GSSB file* and the *cluster ULS file*.

KDCFILE

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

KDCROOT

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

KDCS message area

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

KDCS parameter area

See *parameter area.*

KDCS program interface

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

Kerberos

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

Kerberos principal

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

key code

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

key set

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

linkage program

See *KDCROOT*.

local secondary storage area

See *secondary storage area.*

Log4j

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

lock code

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

logging process

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

LPAP bundle

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

LPAP partner

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

LTERM bundle

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

LTERM group

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

LTERM partner

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

LTERM pool

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

LU6.1

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

LU6.1-LPAP bundle

LPAP bundle for *LU6.1* partner applications.

LU6.1 partner

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

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

main process (Unix /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*.

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.

19 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

20 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*