English

Fujitsu Software BS2000

openUTM V7.0

Using UTM Applications on BS2000 Systems

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

Using UTM Applications on BS2000 Systems

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.

1.1 Summary of contents and target group

This manual is aimed at UTM application planners, application developers, users, and support personnel.

It contains all the information you will need to create a UTM application program on BS2000 systems and implement a UTM application.

The first chapters of this manual provide an overview of how to structure and link a UTM application, and specify which files are needed to operate an application. Separate chapters deal with starting and stopping a UTM application, and with exchanging programs while the application is running. Special issues that you have to take into account when operating a UTM database application are dealt with centrally in separate section with corresponding name.

Full details are provided on how terminal users and other clients can sign on to a UTM application.

There is also a separate chapter describing the tools available for running and controlling a UTM application.

Knowledge of the operating system is a prerequisite.

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.

1.4 Notational conventions

Metasyntax

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

Representation	Meaning	Example
UPPERCASE LETTERS	Uppercase letters denote constants (names of calls, statements, field names, commands and operands etc.) that are to be entered in this format.	`LOAD-MODE=STARTUP`
lowercase letters	In syntax diagrams and operand descriptions, lowercase letters are used to denote place-holders for the operand values.	`KDCFILE=filebase`
lowercase letters in italics	In running text, variables and the names of data structures and fields are indicated by lowercase letters in italics.	*utm-installationpath* is the UTM installation directory
`Typewriter font`	Typewriter font (Courier) is used in running text to identify commands, file names, messages and examples that must be entered in exactly this form or which always have exactly this name or form.	The call `tpcall`
{ } and |	Curly brackets contain alternative entries, of which you must choose one. The individual alternatives are separated within the curly brackets by pipe characters.	`STATUS={ ON | OFF }`
[ ]	Square brackets contain optional entries that can also be omitted.	`KDCFILE=( filebase [, { SINGLE | DOUBLE } ] )`
( )	Where a list of parameters can be specified for an operand, the individual parameters are to be listed in parentheses and separated by commas. If only one parameter is actually specified, you can omit the parentheses.	`KEYS=(key1, key2,...key`*n*`)`
Underscoring	Underscoring denotes the default value.	`CONNECT= { YES | NO }`
abbreviated form	The standard abbreviated form of statements, operands and operand values is emphasized in boldface type. The abbreviated form can be entered in place of the full designation.	**T**`RANSPORT-SEL ECTOR=c‘C‘`

. . .	An ellipsis indicates that a syntactical unit can be repeated. It can also be used to indicate sections of a program or syntax description etc.	`Start KDCDEF ...`
		`OPTION`
		`DATA=statement_file ...`
		`END`

Symbols

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

Indicates notes that are of particular importance.

Indicates warnings.

Other

utmpath	On Unix, Linux and Windows systems, designates the directory under which openUTM was installed.
filebase	On Unix, Linux and Windows systems, designates the directory of the UTM application. This is the base name generated in the KDCDEF statement MAX KDCFILE=.
$userid	On BS2000 systems, designates the user ID under which openUTM was installed.
upic_dir	The directory under which UPIC Client for UPIC Carrier System is installed on Unix, Linux, or Windows system.

2 Defining the application program structure

This chapter discusses the structuring of an application program.

Note that you must link the application program using the BINDER utility and load it using the functions provided by BLS.

A UTM application can be structured in different forms and loaded in various ways. In general, a distinction can be made between five different areas:

- Shareable programs in the system memory

  Program components loaded in the system memory are available to all processes of a BS2000 system in common. For this reason, you should primarily load application-independent programs into the system memory, such as the shareable parts of runtime systems of the programming languages. See also section “Shared code in system memory”.

  Modules must be loaded as a nonprivileged subsystem by the system administrator.

- Statically linked program components

  The ROOT system modules and the runtime system modules they require (ILCS), as well as the message modules for which no dynamic loading library was specified in the UTM generation, must be incorporated statically in the start LLM. The ROOT table module created by KDCDEF and assembled by you can, however, be linked statically or be loaded dynamically at the start.

  Other parts of the application can also be incorporated in the static part, but these must not contain any external references to modules you load into a common memory pool managed by openUTM.

  The static part of a UTM application is loaded with the command:

  `START-EXECUTABLE-PROGRAM` or `LOAD-EXECUTABLE-PROGRAM`

- Shareable programs in common memory pools

  Program components that can be shared by all processes of a UTM application, such as the shareable parts of your program units of the UTM application or formats or data areas, should be loaded in common memory pools. See also section “Shared code in common memory pools”.

- Program components to be loaded when the application starts

  Program components which are always required by the UTM application or which contain external references to shareable parts of the application, should be loaded dynamically at the application start.

  The ROOT table module created and assembled by KDCDEF is also dynamically loaded if you did not link it statically to the start LLM.

- Program components to be loaded when the program is called

  Program components that are not always required by the UTM application can be generated such that they are not loaded dynamically until the first call is issued. These program components must be statically linked to LLMs in such a way that all open external references can be resolved from the modules already in the memory when loading.

The values you specified when generating the application with KDCDEF (see openUTM manual “Generating Applications”) determine the memory areas into which the program components are loaded and the time at which they are loaded.

For more information, please refer to the following chapters:

- chapter “Creating the application program”
- chapter “Exchanging programs during operation”

2.1 Generating load modules

Only part of the application need to be linked statically to the application program (Start-LLM, see section “Linking the application program”). The other parts of the application must then be made available in the form of dynamically reloadable load modules.

These load modules can be individual object modules (OM), which are provided in an object module library (OML) or a program library as type R elements, or link load modules (LLM) generated with BINDER, which are provided in a program library as type L elements. Groups of objects can be statically linked to LLMs using BINDER, see section “Linking LLMs”.

During KDCDEF generation, the individual load modules of the application must be generated with LOAD-MODULE statements. The assignment of objects (program units and shareable data areas) to load modules is likewise defined in the UTM generation (operand LOAD-MODULE in the PROGRAM and AREA statements).

You must define the following for the load modules during KDCDEF generation:

- When the load modules are to be loaded

  For program units, i.e. modules generated with the PROGRAM statement, you can choose between the start time of the application and the call time of the program unit.

  Non-shareable program units can be linked statically to the application program, loaded dynamically as a load module at the application start, or loaded as a load module at the time the program unit is called.

  The non-shareable part (private slice) of a shareable part (public slice) contained in a common memory pool can likewise either be loaded dynamically at the application start or not loaded until the program unit is called.

  The private slice of a public slice contained in nonprivileged subsystems can alternately be linked to the static part of the application program.

  Modules and data areas generated with the AREA statement must either be linked statically to the application program or not loaded dynamically until the application starts, because access to these application parts is not controlled by openUTM.
- The libraries from which the load modules are to be loaded

  You can specify object module libraries (OML) or program libraries (PL) which contain type R or L elements.
- Which version of a load module is to be loaded

  A program library can contain several versions of an element at the same time. You use the version specification to define which version of an element is to be loaded.
- in which storage area the load modules are to be loaded

  A load module can either be loaded in a common memory pool or in the standard context of the application. The standard context contains the statically linked part of the application program, which was loaded with the command START-EXECUTABLE-PROGRAM (LOAD-EXECUTABLE-PROGRAM), as well as the parts of the application program that were not loaded in common memory pools or in the system memory.

  If an LLM contains public and private slices, the public slice can be loaded in a common memory pool and the private slice can be loaded in the standard context in the local task memory.

- Whether autolink is to be used for linking

  The shareable parts of the load module are always loaded without using the Autolink function. You can control whether or not the Autolink function is to be used for loading with the LOAD-MODULE statement.

  Suppressing the autolink function of BLS when loading and when exchanging programs speeds up the load procedure for load modules and avoids inconsistencies in the loaded application program. In this case, the load modules must only have open external references to program components that already exist in the working memory when this module is loaded.

  If autolink is used for loading, the additionally required modules are linked when the load module is loaded.The autolink function should only be used for modules of the runtime system but must *not* be used for user-specific modules because modules loaded with autolink are not unloaded in a subsequent exchange.

The assignment of objects (PROGRAM, AREA statement) to load modules (LOAD-MODULE statement) is also defined in the UTM generation. openUTM cannot verify whether the assignment defined with KDCDEF corresponds to the actual division of load modules in the libraries. In a program exchange, openUTM relies on the specifications made in the UTM generation.

The sequence with which you generate the load modules determines the sequence in which the load modules are loaded. If load modules generated with LOAD-MODE=(POOL, poolname, STARTUP) or LOAD-MODE=STARTUP are loaded without the BLS autolink function, the generation sequence is decisive for the resolution of unresolved external references when loading.

Detailed information on generating the load modules can be found in the openUTM manual “Generating Applications”.

Notes on program exchange

When generating the load modules, you define which program components can be exchanged at a later stage. The following must be noted here:

- Program components that can subsequently be exchanged must not be statically linked into the application program nor loaded into global application common memory pools or the system memory.
- BLS only supports the unloading of program components that were loaded in **one** load procedure. For example, if you want to exchange a group of modules, you must provide these modules as an LLM or OM. If you want to exchange a single module, this module can be provided as an LLM or OM and then cannot be linked with other modules.
- A load module that contains the KDCADM administration program, the event exits START, SHUT, INPUT and FORMAT, or the event services BADTAC, MSGTAC and SIGNON, must not be exchanged during operation.
- In an update generation, the current version numbers of the exchanged load modules are by default transferred from the old KDCFILE to the new KDCFILE if necessary.

2.2 Loading modules

The sequence in which the various program components of the UTM application are loaded and how the external references are resolved are illustrated in figure 1 and described in the following section. The numbers indicate the sequence in which the program components are / must be loaded. The arrows specify the direction in which the unresolved external references of the load modules are resolved when loading, if the autolink function is not used.

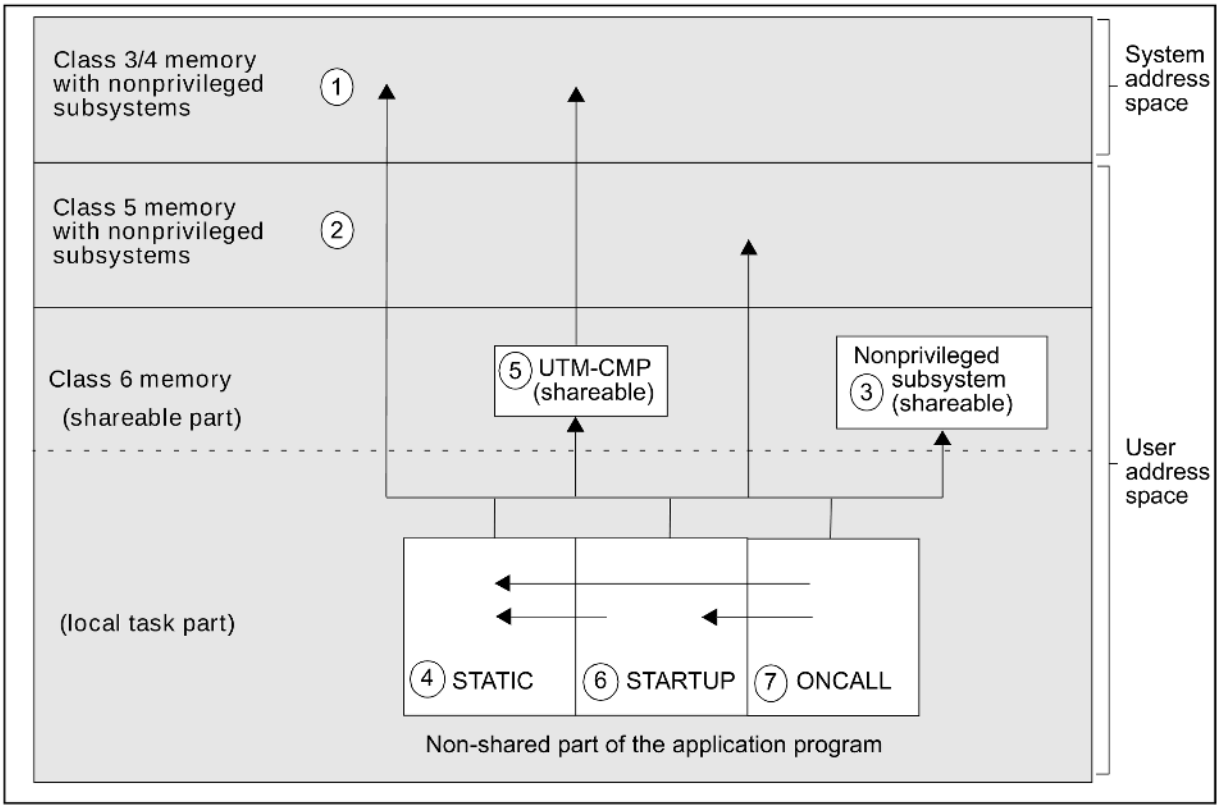


Figure 1: Memory structure of a UTM application (CMP = Common Memory Pool)

All load procedures are initiated and controlled by openUTM with the exception of loading the subsystems (performed by the system administrator) and loading the static part (initiated by a START-EXECUTABLE-PROGRAM command):

1,2,3

1,2,3

Shared modules are loaded by the system administrator before the start of the UTM application.

4

The start LLM with all linked load modules is loaded with the START-EXECUTABLE-PROGRAM or LOAD-EXECUTABLE-PROGRAM command.

Due to the load performance, open external references should only be resolved from the shared code, but can also be resolved from libraries.

The command for starting the start LLM should be as follows:

*) or LOAD-EXECUTABLE-PROGRAM

The parameters UNRESOLVED-EXTRNS=*DELAY and LOAD-INFORMATION=*REFERENCES are mandatory for starting UTM applications.
AUTOLINK=*ALTERNATE-LIBRARIES specifies that only the alternative libraries are to be used for the autolink function, and ALT-LIB=*BLSLIB## specifies that only libraries
with the link names BLSLIBnn can be used as alternative libraries.

```
/START-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM                        - *)
/     (LIBRARY=lm-lib                                                -
/     ,ELEMENT=lm-name, TYPE=L)                                      -
/     ,DBL-PAR=(LOADING=                                             -
/                 (LOAD-INFORMATION=REFERENCES,PROGRAM-MODE=ANY)    -
/              ,RESOLUTION=(ALTERNATE-LIBRARIES=*BLSLIB##           -
/                          ,AUTOLINK=*ALTERNATE-LIBRARIES)          -
/              ,ERROR-PROCESSING=(UNRESOLVED-EXTRNS=*DELAY))
```

The command for starting the start LLM should be as follows:

*) or LOAD-EXECUTABLE-PROGRAM

The parameters UNRESOLVED-EXTRNS=*DELAY and LOAD-INFORMATION=*REFERENCES are mandatory for starting UTM applications.

AUTOLINK=*ALTERNATE-LIBRARIES specifies that only the alternative libraries are to be used for the autolink function, and ALT-LIB=*BLSLIB## specifies that only libraries
with the link names BLSLIBnn can be used as alternative libraries.

5 All load modules generated with LOAD-MODE=POOL and with a common memory pool with SCOPE=GLOBAL are loaded in the sequence of the MPOOL statements.
After that, all load modules that were generated with LOAD-MODULE=POOL and with a common memory pool with SCOPE=GROUP are loaded.
Within a pool, the load modules are loaded into the pool in the order specified in the LOAD-MODULE statement.
When loading, only the external references from system memory and from their own memory pools are resolved; the Autolink function is suppressed when loading shared
code.

6 The load modules generated with LOAD-MODE=STARTUP are loaded dynamically at the start of the application program. The sequence of the LOAD-MODULE statements in the UTM generation determines the sequence when loading.

Open external references can be resolved from the system memory, from the common memory pools and from the modules already loaded. External references to the runtime systems can be resolved by dynamically loading the runtime system modules (specify the operand ALTERNATE-LIBRARIES=YES in the LOAD-MODULE statement).

The same is true when loading the private slices of load modules generated with LOAD-MODE=(POOL,..., STARTUP | ONCALL).

7 Load modules generated with LOAD-MODE=ONCALL are loaded the first time an associated program unit is called. Open external references are resolved as for load modules generated with LOAD-MODE=STARTUP (see 6).

2.3 Recommendations for structuring the application

The load structure of an application definitively determines the time required to start the first task and the follow-up tasks of an application. It also defines which program components can be exchanged at a later stage in the application run. For this reason, you should note the following recommendations:

- The statically linked part of the application program to be loaded with the START-EXECUTABLE-PROGRAM command (start LLM) should be kept as small as possible. It must contain the ROOT system modules and the runtime modules required by the system modules of KDCROOT.
- The administration modules (e.g the KDCADM administration program) are to be statically linked to the start LLM or must be loaded when the application is started, i.e. it must be generated with LOAD-MODULE ...,LOAD-MODE=STARTUP. Self-written administration programs should be loaded like the loading module KDCADM when starting the application, i.e. they should be statically bound to the start LLM like the loading module KDCADM or loaded when starting the application. A load module containing the administration program KDCADM must not be exchanged during operation. Therefore you must not connect self-written administration programs, which should be exchangeable, with the KDCADM.
- The START, SHUT, INPUT and FORMAT event exits and the BADTAC, MSGTAC and SIGNON event services are to be statically linked to the start LLM or to one of their own load modules. This load module must be loaded when the application is started, i.e. it must be generated with LOAD-MODULE ...,LOAD-MODE=STARTUP. It must not be exchanged during operation because errors occurring during the exchange can lead to the abnormal termination of the application.
- Program components that are to be exchanged at a later stage must not be linked statically or loaded into global application common memory pools or into the system memory.
- Program components that are required by more than one application should be shareable, e.g. should be loaded as a nonprivileged subsystem.
- All other application parts that are shareable should be pre-linked to form a load module that should be loaded into a common memory pool. If several load modules with shareable program components are required for some reason, these should nevertheless be loaded into only **one** common memory pool.
- Many compilers allow you to save the shareable and the non-shareable module together in one LLM. This simplifies the administration of the modules, as both parts are always located in one container. In this case, the shareable and non-shareable parts should not be statically linked separately, rather the LLMs with public and private slices, as created by the compiler, should be statically linked in one or more LLMs.

  openUTM loads the public slice of this type of LLM into a memory pool and loads the private slice into the local task memory at the application start or when a program from the LLM is called, if you generate the LLM with LOAD-MODULE ..., LOAD-MODE=(POOL,...,STARTUP) or LOAD-MODE= (POOL,...,ONCALL).
- Program components that are only required occasionally should be generated such that they are not loaded dynamically until they are called (LOAD-MODULE ...,LOAD-MODE=ONCALL).
  This shortens the start phase. These load modules should be statically linked in small logical units, because it takes less time to load small modules that large ones.
- Application parts that are logically related should be statically linked in a load module. These may be program components that refer to each other or which belong to a sequence of program unit runs in a service.
- All program components that are to be loaded dynamically should be statically linked as LLM. This considerably speeds up the start of the application and the loading of load modules, because the dynamic binder loader (DBL) can process LLMs much more efficiently than OMs.

- The number of load modules loaded when the application program starts (LOAD-MODE=STARTUP or LOAD-MODE=(POOL,...,STARTUP) should be kept as small as possible. For this reason, only the parts of the application that are frequently required should be generated with STARTUP.
  Since it takes less time to load a small number of large load modules than a large number of small load modules, these parts of the application program should be statically linked to larger load modules.
- Only **one** local application common memory pool should be defined (MPOOL ..., SCOPE=GROUP). A number of statically linked load modules can be loaded into this pool. This reduces the time required to set up and load the common memory pools and thereby minimizes the time needed to start the application.
- OMs and LLMs should be made available in different program libraries because this optimizes the BLS search algorithm.
- Modules that are neither program units of the application program nor data areas (AREA) (e.g. the modules of the runtime systems of the programming languages) need not be declared as dynamically loaded modules with the KDCDEF generation tool, even if these modules are not linked statically. You can statically link these modules to larger load modules (LLM) and need only generate the name of the load module in the LOAD-MODULE statement.

CAUTION!

openUTM cannot verify whether the assignment defined with the LOAD-MODULE statement and the LOAD-MODULE operand in the PROGRAM and AREA statements corresponds to the actual division of the load modules in the libraries. When dynamically loading the load modules, openUTM relies on the specifications made in the UTM generation. You must therefore ensure that the link procedures you use for the individual parts of the application program correspond with the specifications made in the UTM generation. Otherwise, openUTM cannot guarantee that a required program will be loaded in the working memory with a particular load module.

2.4 Rules and restrictions

The following rules and restrictions must be observed when dividing objects into load modules and when linking and dynamically loading objects:

- The event exits START, SHUT, INPUT and FORMAT and the event services BADTAC, MSGTAC and SIGNON must not be assigned to any load module whose loading is deferred until a program unit is called, i.e. is generated with LOAD-MODE=ONCALL. The reason for this is that the event exits and event services must be available in each process of the application. If they are not, openUTM aborts the start of the process.The load modules that contain event exits and event services must not be exchanged during operation. Errors occurring during the exchange of the load modules can lead to the abnormal termination of the application.
- The administration program KDCADM must not be assigned to any load module generated with LOAD-MODE=ONCALL.
  The load module to which the KDCADM administration program is linked must not be exchanged.
  If the administration program KDCADM is not available when the application is started, openUTM aborts the start.
- Data areas (AREAs) must either be statically linked to the application program, or, if possible, generated as LOAD-MODULEs with LOAD-MODE(POOL, STARTUP) or LOAD-MODE=STARTUP.
- The start LLM must be made available in a program library. It must not be contained in a file.
- You must specify the following operands in the START-EXECUTABLE-PROGRAM command:

```
/     ,DBL-PARAMETERS=*PARAMETERS(                                  -
/            ,LOADING=*PARAMETERS(                                  -
/                          PROGRAM-MODE=*ANY                        -
/                         ,LOAD-INFORMATION=*REFERENCES)            -
...
/          ,ERROR-PROCESSING=*PARAMETERS(                           -
/                          UNRESOLVED-EXTRNS=*DELAY                 -
/                         ,...))
```

  You must specify PROGRAM-MODE=*ANY if the application is to be loaded into the upper address space.
- If an administration command requires a program exchange, you must explicitly specify the version of the new module to be loaded.
- For load modules, you must only specify the names of OMs or LLMs. For performance reasons, openUTM does not support dynamic loading using CSECT or ENTRY names.
- openUTM supports a maximum of eight common memory pools with SCOPE=GROUP and eight common memory pools with SCOPE=GLOBAL.
- Two UTM applications should be started under different BS2000 user IDs if possible to prevent errors that can arise due to having the same module name appear twice in a shareable section. Modules that are used by several applications are therefore to be loaded into global common memory pools or in nonprivileged subsystems.
- A newly created table module can be created by exchanging a program.
- The variant number of the LMS is of no significance when loading load modules with BLS.
- BLS only supports the unloading of program components that were loaded in **one** load procedure. For this reason, program components that are to be exchangeable as separate parts must be made available in the form in which they are to be loaded and exchanged. If an individual module is to be exchanged, this module can be made available as an OM or LLM; if a group of load modules is to be exchanged, these modules must be statically linked as LLMs.

- Load modules containing TCB entries must not be exchanged during operation.
- Modules that are loaded dynamically using the Autolink function cannot be exchanged or unloaded. Exception: When exchanging the entire application (e.g. with KDCAPPL PROG=NEW), *all* parts of the application are reloaded.
- FHS can not be loaded from the TASKLIB. FHS format modules can be exchanged with this version or later.
- When statically linking load modules to LLMs, please note that the incorporated elements should not appear in more than one load module. Restrictions therefore apply, particularly when incorporating the runtime systems of the programming languages. For information on how to link the runtime system modules required for the application program, see section “Linking runtime systems”.

2.5 Using shared code

Many compilers offer the option of creating a shareable part when compiling programs. This shareable part need not necessarily be saved in a separate object module, rather can be contained with the non-shareable part in an LLM, which is subdivided into a public and a private slice.

The following objects can be loaded as shareable:

- shareable modules of program units
- formats
- some of the modules for formatting (if shareable)
- shareable data areas
- the database connection module, if this is shareable
- the message modules
- modules of the runtime systems of the programming languages that are shareable (see the manuals of the individual programming languages)

If parts of a program unit are to be shareable, this must be taken into account in the programming. The compiler which compiles the program unit determines what you must note when programming. For further information, see the openUTM manual „Programming Applications with KDCS” or the appropriate language supplement.

> ¡ COBOL compiler refers either to COBOL85 or to COBOL2000.

COBOL program units compiled with the COBOL85 compiler are shareable if you set the option `COMOPT GEN-SHARE-CODE=YES` when compiling. Specifying the option `COMOPT GEN-LLM=YES` instructs the compiler to save the objects in an LLM with slices.

COBOL program units compiled using the COBOL2000 compiler are shareableif you specify the following options during compilation:

```
COMPILER-ACTION=*MOD-GEN(SHAREABLE-CODE=*YES, MODULE-FORMAT=*LLM,...)
```

C/C++ program units are shareable if you specify the following option when compiling:

```
COMPILER-ACTION=MODULE-GENERATION( SHAREABLE-CODE=YES,...)
```

Please refer to the relevant user guides for the corresponding option in other programming languages.

In addition, many runtime systems of the programming languages are shipped with a shareable part.

> ¡ if you wish to exchange a program unit that is made up of a non-shareable module and a shareable module, then you must link the shareable and non-shareable parts to an LLM because otherwise inconsistencies would arise in the exchange. You can avoid consistency problems by locking the TACs contained in the program unit before the exchange, and then release them again after the shareable and non-shareable parts have been exchanged.

Loading shared code

The shareable program components need only be loaded once together for all tasks of the application(s). Only the non-shareable parts need then be loaded in the local task memory.

You have various options for loading shareable objects:

- as a nonprivileged subsystem in the class 3/4/5 or 6 memory in the **system storage**
- in a **common memory pool** managed by openUTM in the user storage area(class 6 memory)

See also the “BS2000/OSD Subsystem Management (DSSM/SSCM)” manual.

2.5.1 Shared code in system memory

Using the interfaces provided on BS2000 systems, shareable parts of the application program units and parts of the runtime systems can be loaded either as shareable programs in nonprivileged subsystems.

Shared code loaded in nonprivileged subsystems can be exchanged while the application is running.

The **shareable** modules must be loaded in the memory by the administrator before the application is started. See also the notes on the following page. In contrast to shared code in common memory pools which can be loaded and exchanged under the control of openUTM, you cannot manage the shareable modules in system memory yourself.

Non-shareable parts of the program units must be described with the PROGRAM statement and, if necessary, assigned to a load module with the LOAD-MODULE statement.
The load module or its private slice is loaded dynamically into the local task memory (class 6 memory) when the application program is started. The links are established dynamically in the shared code using the external references to shareable modules.
The load modules (OM format) that contain the shareable modules of the program unit need not reside in the same program library as the load modules that contain the non-shareable program components.

Notes

- Subsystems should be created in LLM format because these subsystems can be exchanged **without** inconsistencies while the application is running.
  The public slice of the old LLM may eventually be loaded in the local task part of the application program when exchanging the subsystem until the private slice is also exchanged (e.g. initialized with KDCPROG).

  On the other hand, when subsystems in OM format are exchanged during operation, an inconsistency exists between the DSSM subsystem command

  ```
  START-SUBSYSTEM SUBSYSTEM-NAME=subsystemname -
                , VERSION=new-version         -
                , VERSION-PARALLELISM=*EXCHANGE-MODE
  ```

  and the UTM administration command (or a corresponding KDCADMI call in the administration program interface)

  `KDCPROG LOAD-MODULE=load-module,VERSION=new-version`

  It may happen in this case that after the DSSM command and before the UTM administration command, a UTM application program is terminated with PEND ER and the affected task of the UTM application program is started with an old private part and a new public part in a new subsystem .
- When exchanging subsystems of various versions whose modules are contained in the same library (condition for exchanging load modules with openUTM), please make sure that different LINK-ENTRYs are specified as otherwise the DSSM may not be able to implement the exchange.
- When creating a subsystem, the corresponding LLM format (e.g. Format 2) should be used because in this case only a subsystem entry need be specified. See also the “BINDER User Guide”.

2.5.2 Shared code in common memory pools

A common memory pool is an area in the user storage area (class 6 memory). Shareable objects that are not linked statically when linking the application program can be loaded into a common memory pool. openUTM offers two options for setting up a common memory pool:

- As a local application pool

  All program units of an application can access this pool. Program units of other applications can also access the pool provided these applications were started under the same BS2000 user ID.

- As a global application pool

  Program units of **several** applications can access the same common memory pool, regardless of the user IDs under which these applications were started.

Each common memory pool must be generated with the KDCDEF statement MPOOL. In the MPOOL statement you define whether a common memory pool is to be set up as a global or local application pool.

2.5.2.1 Local application pool

A local application pool is created by the first task of an application. All subsequent tasks of this application likewise access this pool, i.e. the pool is created with SCOPE=GROUP in the context of BS2000 memory pool management. Tasks of other UTM applications that were started under the same user ID can also connect to this common memory pool.

The local application pool offers the following advantages for the user:

- The programs loaded in this pool can only be used from this user ID.
- The user has full control over the use of shareable objects, regardless of the system initialization.
- The pool only exists for as long as the application is running or, if a number of applications are accessing the pool, until the last application has terminated. When the application ends, the pool is cleared and the programs – like all other application programs – are unloaded.
- The programs loaded in a common memory pool can also be exchanged dynamically during the application run under certain conditions. For more information, see the chapter “Exchanging programs during operation”.
- The physical memory is utilized more efficiently.

When the (last) application terminates, the pool is cleared and the programs are unloaded (as are all other application programs).

> **CAUTION!**
>
> All modules, formats, and data areas have the same access authorization within a pool.
>
> Programs loaded in common memory pools may not contain any open external references to addresses outside the common memory pool which point to modules located in other common memory pools or nonprivileged subsystems in class 5/6 memory. These CSECTs and ENTRYs are not found. Moreover, open external references must only point to shared code located at the same address for all tasks. See also the DSSM manuals.

2.5.2.2 Global application pool

A global application pool is used for modules that are needed in several applications, e.g. formats, connection module for formatting, database connection module, language runtime modules, shareable data areas, and also routines in program units and the message module.

A global application pool is set up by the first task of the first UTM application. All subsequent tasks of the same and other applications can connect to this pool, i.e. the pool is created with SCOPE=GLOBAL in the context of BS2000 system‘s memory pool management.

A global application pool offers similar advantages to a local application pool; however, the disadvantage is that the programs loaded in a global application pool cannot be exchanged dynamically while the application is running.

If global common memory pools of the same content/name are used in several UTM applications, the PAGE=X’ xxxxxxxx’ parameter in the KDCDEF statement MPOOL must be specified with the same address in all applications. The address specified with PAGE= must be chosen such that the reserved address area is available in all of these applications.

Example

Application 1 uses the global pool MPONE. Application 2 also uses this pool in addition to pool MPTWO. The following KDCDEF control statements are required:

UTM generation application 1:

```
MPOOL MPONE, SCOPE=GLOBAL, PAGE=X’01000000’, SIZE=...
```

UTM generation application 2:

```
MPOOL MPTWO, SCOPE=GLOBAL, SIZE=...

MPOOL MPONE, SCOPE=GLOBAL, PAGE=X’01000000’, SIZE=...
```

As an alternative to using PAGE=, the shared pools can be generated in the same sequence in all applications. Moreover, these MPOOL statements must be the first MPOOL statements to be specified.

2.5.2.3 Generating shareable objects that are loaded in a common memory pool

If parts of your application program are to be loaded in a common memory pool, you should note the following:

- For performance reasons, all shareable parts of an application program that are to be loaded in a common memory pool should, as far as possible, be combined into **one** load module.
- The program's shareable code module created by the compiler must be contained in an LLM or OM.
- If a compiler created two separate object modules for the shareable and non-shareable part, then you should link these modules beforehand to an LLM with slices using the binder.
- LLMs with slices can be generated with a single LOAD-MODULE statement:

```
LOAD-MODULE llm-name ,VERSION=version       -
,LOAD-MODE=(POOL,poolname,{STARTUP|ONCALL}) -
,LIB=program-lib                            -
.ALTERNATE-LIBRARIES={YES|NO}
```

  With this statement, the public slice of the LLM is loaded in the common memory pool *poolname*, while the private slice is loaded later either when the application is started (STARTUP) or when the program is called (ONCALL). PROGRAM statements are also needed for the programs of this LLM that are to be called using openUTM.

  Alternately, you can also generate the shareable and non-shareable module using two LOAD-MODULE statements. You should avoid this, if possible, because you cannot exchange these two modules without having inconsistencies arise.

- A shareable data area that is to be loaded in the common memory pool must be described with an AREA statement. The area must then be contained in the load module, which is generated as follows:

```
LOAD-MODULE ar-share ,VERSION=version                              -
,                    LOAD-MODE=(POOL,poolname,NO-PRIVATE-SLICE)    -
,                    LIB=libname
```

  AREAs that were assigned the PUBLIC attribute during compilation or by the binder can also be linked together beforehand with other modules in one LLM with slices. This LLM can then be generated as follows:

```
LOAD-MODULE llm-with-slices ,VERSION=version                       -
,                           LOAD-MODE=(POOL,poolname,{STARTUP|ONCALL}) -
,                           LIB=libname
```

Example

The example assumes that the COBOL compiler was used for compiling and that the compiler stored the objects in an LLM. The shareable modules of the COBOL program units PU1 and PU2, the format descriptions FORMAT1 and FORMAT2, and the data module DATAMOD are to be loaded in the local application pool LCPOOL. LCPOOL is to be loaded at address X'020000', is to be capable of occupying 128 KB, and is to be write-protected.

```
/MPOOL        LCPOOL,SIZE=2,SCOPE=GROUP,ACCESS=READ,PAGE=X'20000'
/LOAD-MODULE  LLM-LCPOOL,VERSION=1, -
/             LOAD-MODE=(POOL,LCPOOL,STARTUP), -
/             LIB=libname
/PROGRAM      PU1 ,LOAD-MODULE=LLM-LCPOOL,COMP=ILCS
/PROGRAM      PU2 ,LOAD-MODULE=LLM-LCPOOL,COMP=ILCS
/AREA         DATAMOD,LOAD-MODULE=LLM-LCPOOL
```

The object modules must be prelinked to the LLM LLM-LCPOOL before the application starts, which means that the option BY-ATTRIBUTES(PUBLIC=YES) must be specified in the BINDER statement START-LLM-CREATION, whereby the LLM is split into a public slice and a private slice. The LLM created in this way must be supplied in the *libname* library.

3 Creating the application program

Before starting the application, you must create and compile program units. The program units implement the application logic. Further information can be found in the openUTM manual „Programming Applications with KDCS”.

To ensure that the program units will run under openUTM, the UTM application program must be created as follows:

- assemble the ROOT table source generated by the KDCDEF generation tool
- link the ROOT system modules and the language-specific runtime systems to a start LLM; you can also incorporate other application-specific components, such as UTM message modules, format libraries, and program units into the start LLM

The ROOT table module can be loaded dynamically when starting the application.

The diagram below shows the individual steps involved in generating a UTM application program.

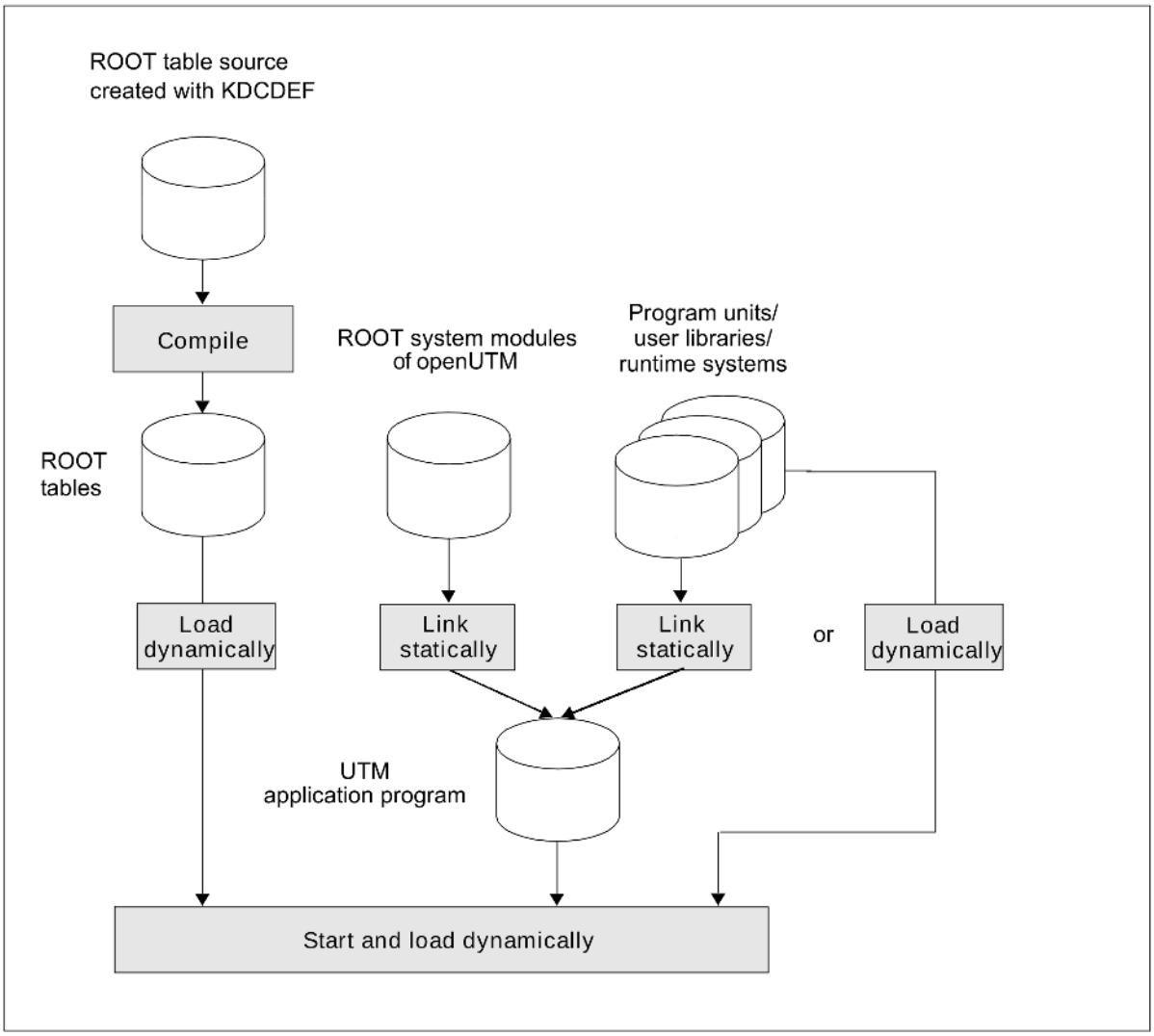


Figure 2: Overview: generating and starting the UTM application program with dynamic loading

The same as other parts of the application, the ROOT tables can also be incorporated into the application program statically, as shown in the diagram below.

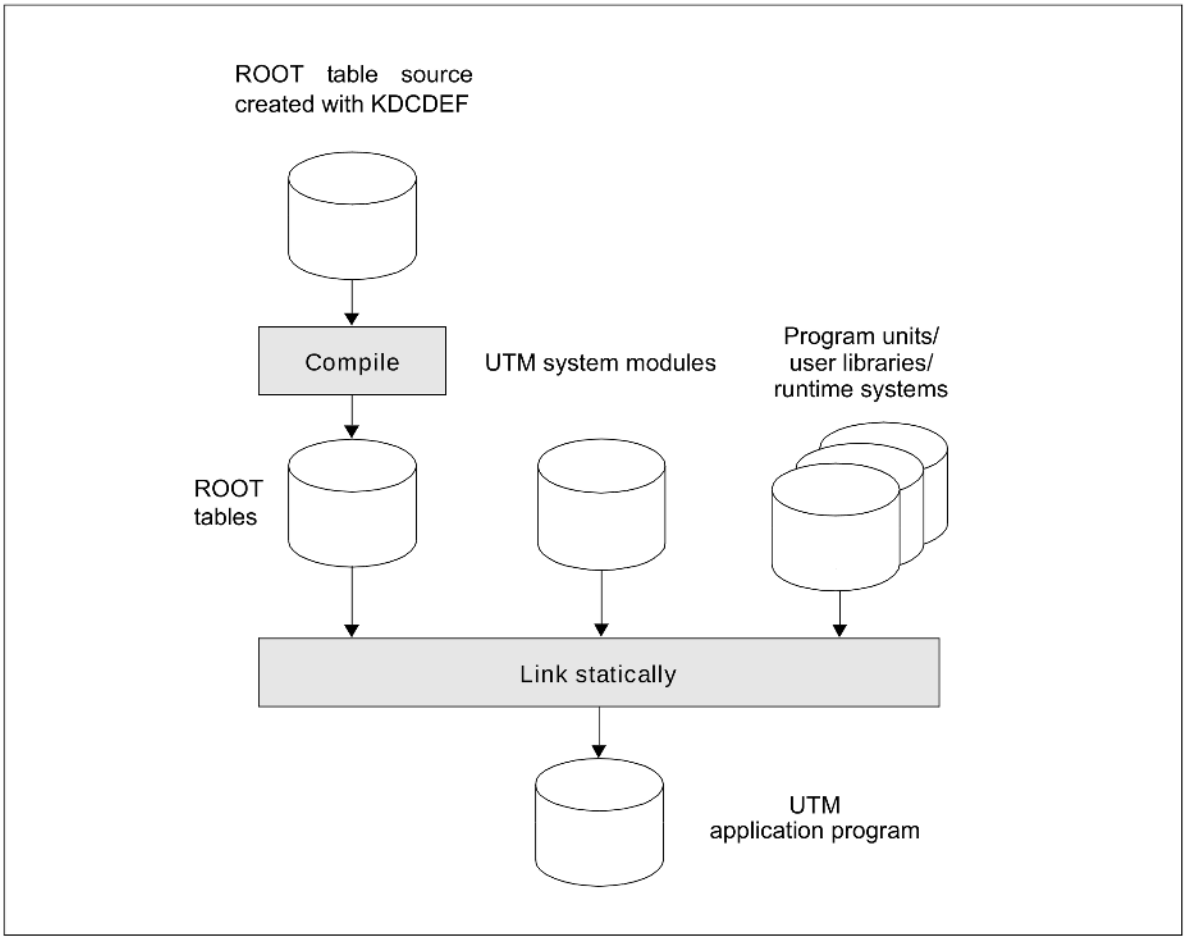


Figure 3: Overview: generating the UTM application program without dynamic loading

Main routine KDCROOT

Based on the ROOT system modules supplied with openUTM, the main routine KDCROOT is generated as part of the application program during the link procedure. When running the application, KDCROOT acts as the main control program. Its tasks include:

- linking program units and UTM system functions
- coordinating the execution of program units in different programming languages
- connecting to databases
- interacting with format handling systems

KDCROOT also contains the variable data and message areas. Further information on the main routine KDCROOT can be found in the openUTM manual „Programming Applications with KDCS”.

The default name for the ROOT table source is ROOT.SRC.ASSEMB.*rootname*. The source for the ROOT tables generated by KDCDEF must be assembled, e.g. using ASSEMBH-GEN. The ROOT system modules associated with the application program are stored in the SYSLNK.UTM.070 module library.

3.1 Components of the application program

A UTM application program is made up of a set of modules which must be linked as a program at runtime or before.

The following modules are required for execution:

- root module

  Compiled ROOT table module which you must store in a program library or object module library. If you use the root dynamic loading technique, specify this library in the start parameter TABLIB= and the module name in the start parameter ROOTNAME=.

- ROOT system modules

  Modules required by the UTM application for execution. They are contained in the library SYSLNK.UTM.070.

- Program units

  The application program units you created which must be compiled and entered in one or more program libraries or object module libraries.

- Database connection module

  Used for connecting the UTM application to the database system specified in the TYPE operand of the DATABASE statement during KDCDEF generation. The manuals for the corresponding database systems describe how this module is provided by the database system.

- Connection module for the format handling system

  Used for connecting the UTM application to the format handling system generated in the TYPE operand of the FORMSYS statement. The manuals for the corresponding format handling systems describe how this module is provided by the format handling system.

- Administration and UTM-SAT administration program unit

  Programs for administering the UTM application. You can either write this type of program yourself using the program interface for administration, or you can use the standard administration programs KDCADM and KDCSADM supplied with openUTM. Each administration program must be generated in a separate PROGRAM statement. The object modules of the administration programs are available in the library SYSLNK.UTM.070. The administration program KDCADM is always needed, even if you are using an administration program which you have created yourself.

- Runtime system for KDCROOT

  This is always required and is supplied in the library SYSLNK.UTM.070.SPLRTS. In addition, KDCROOT needs one of the libraries SYSLNK.CRTE or SYSLNK.CRTE.PARTIAL-BIND, whereby the second library provides performance advantages and is therefore the preferred option. CRTE is the software prerequisite for openUTM and contains the C runtime system, the ILCS, and the COBOL runtime system. The C runtime system and the ILCS are always required by UTM.

- Runtime systems of the programming languages

  These are required if at least one program unit of the application is written in a higher programming language. If the application contains ILCS-capable program units, you must ensure that the highest available ILCS version is incorporated. Further linking information can be found in the description of ILCS in the CRTE User Guide. The ILCS modules are contained in the CRTE library.

  The section “Compiler versions, runtime systems, KDCDEF options” describes which runtime systems can be used with which compiler versions.

- Language connection modules

  These are needed for every generated programming language or its linkage. For SPL, Assembler, COB1 and ILCS linkage, these connection modules are already incorporated in the runtime system for KDCROOT. For all other programming languages, the connection modules must be incorporated from language-specific libraries when the application starts or before.

- Shared areas

  (see AREA statement of KDCDEF)

- User-specific message modules

  If user-specific message modules are to be linked statically.

3.2 Linking the application program

UTM applications use the functionality of the BLS to load LLMs (link and load modules) and OMs (object modules). This opens up new possibilities:

- When managing objects using LLMs that are subdivided into a private and a public slice.

  This means that the shareable and non-shareable parts of an object generated in a compiler run can be managed in a container. This largely avoids inconsistencies when pre-linking and replacing objects.
- When loading an object (program, area).
  An object can be incorporated statically or loaded dynamically at the application start. In the case of program units, it is also possible to defer dynamic loading until the first call is executed.
- When exchanging the program.
  An individual object (LLM or OM) or a statically linked group of objects (LLM or OM) can be exchanged.

Only some of the application objects need be linked statically to the application program. This part of the application program is called the **start LLM** and is loaded and started using the command START-EXECUTABLE-PROGRAM FROM-FILE=*LIBRARY(...). You must link in those ROOT system modules and the objects (program units, modules and data areas) in the start LLM that are not assigned to a load modue or are assigned to a load module with LOAD-MODE=STATIC.

The ROOT table module created by KDCDEF need not to be linked statically, i.e. it can be loaded dynamically at the application start.

The other parts of the application program can be made available in the form of load modules. A load module is

- either a statically linked link load module (**LLM**) contained in a program library as a type L element,
- or an object module (**OM**) contained in an object module library as a type R element.

For performance reasons, however, you should use BINDER to statically link all load modules to LLMs and make them available in a program library as type L elements.

When generating, you must enter precisely one LOAD-MODULE statement for each link load module and each object module that you do not link into the START-LLM statically, and thereby specify when the module is to be linked and where it is to be loaded. For more information, see the openUTM manual “Generating Applications”.

If there are unresolved external references to program units when you link the application, you can still start the application. If you cannot resolve these external references by dynamic loading, the assigned transaction codes for the application are invalid, i.e. they cannot be used.

3.2.1 LLMs with slices

Using BINDER, the CSECTs contained in an LLM can be combined to form a loadable unit known as a **slice**, on the basis of certain criteria. A distinction is made here between slices formed using attributes of CSECTs (slices by attributes) and slices defined by the user.

openUTM can only process LLMs with slices by attributes, and thereby supports the PUBLIC attribute. Many compilers offer the option of storing the created objects in an LLM with a public and private slice. The public slice then contains the shareable part of the object while the private slice contains the non-shareable part. This procedure simplifies the management of shareable and non-shareable parts of a program because both are contained in an LLM; see also section “Linking LLMs to public/private slices”. If you generate this type of LLM as appropriate for openUTM, then openUTM loads the public slice in a common memory pool and the private slice in the local task memory.

If the public slice of an LLM cannot be loaded in the common memory pool when loading or exchanging, this task is terminated after an appropriate K078 message has been issued and application exchange is canceled.

3.2.2 Linking LLMs

All module groups in a logical relationship should be statically linked as LLMs. Such module groups include:

- parts of the runtime systems
- logically related parts of a UTM application, e.g. all program units of one or more services including conversation exits
- FHS formats including the associated program units

In order that the external references of an LLM can be resolved when this LLM is dynamically linked, the following BINDER statements must be specified when creating the LLM:

```
/START-LLM-CREATION    INTERNAL-NAME=int-name - ————————————————————  (1)
/                     ,INTERNAL-VERSION=int-vers -
/                     ,...
/SET-EXTERN-RESOLUTION SYMBOL-TYPE=*REFERENCES - ————————————————————  (2)
/                     ,RESOLUTION=*STD
/SAVE-LLM          ...,ELEMENT=*INTERNAL-NAME( - ————————————————————  (3)
/                      VERSION=*INTERNAL-VERSION) -
/                     ,FOR-BS2000-VERSIONS=*FROM-OSD-V4(...)
```

1. Here you define the name and version of the LLM, and how the LLM is saved in the library when linking is concluded.
2. The handling of unresolved external references is defined. The specified operand values are the defaults which must on no account be modified.
3. The LLM must be saved under the internal name; the format of the LLM must be correct (at least LLM format 2 as in the example).

   If a name other than the internal name is selected when saving and if LOAD-MODULE,...,ALTLIB=YES is specified when generating, openUTM *cannot find the module when a program is exchanged and reports an* UNBIND erro*r* (`K078 UNBIND 0C010174`*).*

Repeated elements

When statically linking LLMs, you must ensure that the incorporated elements (sub-LLMs / OMs) do not occur in several (possibly statically linked) LLMs / OMs. If this is the case, however, the external names contained in repeatedly linked modules must be masked by the BINDER statement MODIFY-SYMBOL-VISIBILITY.

Autolink function for dynamically loading load modules

The autolink function is controlled in the LOAD-MODULE statement by the operand ALTERNATE-LIBRARIES.

- ALTERNATE-LIBRARIES=NO
  switches off the autolink function of BLS. This means that for all dynamically generated load modules loaded in this way, all open references of such a load module must be resolved by the modules loaded at load time (start LLM and other load modules) and by the shared code. External references in the runtime system are always resolved for load modules comprising only C or data objects if a RESOLVE-BY-AUTOLINK to the library SYSLNK.CRTE.PARTIAL-BIND was specified when linking the start LLM.

- ALTERNATE-LIBRARIES=YES
  switches on the autolink function of BLS.
  You can use this for load modules which, when loaded and exchanged, require additional modules of the runtime system that are not yet contained in the memory. When linking, open external references are first sought in the link context and then in the shared code (see also the manual „Dynamic Binder Loader/Starter“). If external references are still unresolved after this, the library you specified in the LIB operand is searched. If a suitable definition is found, the associated module is linked and the search is terminated. Otherwise, the search is continued in the libraries to which you have set a link with the SET-FILE-LINK command before the application is started with the link name BLSLIB*nn* (where 00<=*nn*<=99). *The libraries are processed in ascending order of nn.*

  The handling of the runtime system modules in connection with UTM applications is described in detail on "Linking runtime systems".

The autolink mechanism should not be applied on user-specific libraries because only the load module and not the entire load unit, which also contains all modules loaded by autolink, is unloaded when a program is exchanged.

The autolink function for the start LLM can be influenced by the parameters of the command START-EXECUTABLE-PROGAM. See also the example on "Loading modules".

Example

- UTM generation and LINK statement

  UTM generation:

```
LOAD-MODULE     lm-name                                          -
                ,VERSION             = llm-version               -
                ,LIB                 = lm-lib                    -
                ,LOAD-MODE           = ONCALL                    -
                ,ALTERNATE-LIBRARIES = YES
```

  LINK statement before application start:

```
/SET-FILE-LINK  LINK-NAME = BLSLIB00,FILE-NAME = $userid.SYSLNK.CRTE.PARTIAL-BIND
```

  First of all, the LOAD-MODULE statement generates the load module *lm-name* with LOAD-MODE=ONCALL, i.e. *lm-name* is not loaded dynamically until a constituent program is called. In the example, *lm-name* should contain COBOL objects.

  To resolve open external references when loading, the library *lm-lib* specified in the LIB operand is first searched and then the library *$userid*.SYSLNK.CRTE.PARTIAL-BIND, which is assigned using the SET-FILE-LINK command.

- BINDER statements

```
/START-BINDER
//START-LLM-CREATION     INTERNAL-NAME    = llm-name             -
//                      ,INTERNAL-VERSION = llm-version
//INCLUDE-MODULES         LIBRARY=user-library                    -
//                      ,ELEMENT=program-unit-name
//RESOLVE-BY-AUTOLINK     LIBRARY=(library1,library2..)
//SET-EXTERN-RESOLUTION   SYMBOL-TYPE=*REFERENCES                 -
//                      ,RESOLUTION=*STD
//SAVE-LLM                LIBRARY=llm-library                     -
//                      ,ELEMENT=*INTERNAL-NAME(                  -
//                       VERSION=*INTERNAL-VERSION)               -
//                      ,FOR-BS2000-VERSIONS=*FROM-OSD-V4(...)
//END
```

In this binder run, an application program unit is linked to a load module. KDCROOT is not incorporated here, e. g. because the KDCS entry remains unresolved.

3.2.3 Linking LLMs to public/private slices

If you use the linkage editor to link shareable modules in OM format to LLMs with a public slice, you must ensure that the CSECTs of these modules have the attribute PUBLIC. If this is not the case, the linkage editor creates these CSECTs in the private slice.

This can occur for example with AREAs that were previously in shareable modules in OM format (the attribute has no significance here) or with all shareable COBOL objects. You can adapt the objects by

- adding and then recompiling the PUBLIC attribute for AREAs,
- recompiling the associated sources in LLM format for shareable COBOL objects, or
- setting the PUBLIC attribute with the linkage editor.

The following example illustrates the third option.

Example

A load module is to be statically linked as an LLM. It should comprise

- the COBOL objects AFPUT and COBECHO with the non-shareable partsAFPUT and COBECHO and the shareable parts AFPUT@ and COBECHO@
- as well as the shareable AREAs AREA1, AREA2 and AREA3, all of which are missing the PUBLIC attribute.

This results in the following binder statements:

```
//START-LLM-CREATION                                                -
//INTERNAL-NAME  = EXAMPLE-LLM                                      -
//              ,INTERNAL-VERSION    = 001                          -
//              ,SLICE-DEFINITION    = BY-ATTR( PUBLIC = YES )
//REMARK ------------------------------------------------------------
//REMARK ------------------------------------------------------------
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = COBECHO
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = AFPUT
//REMARK ------------------------------------------------------------
//REMARK ------------------------------------------------------------
//BEGIN-SUB-LLM SUB-LLM-NAME=OM-WITHOUT-PUBLIC-ATTRIBUTE
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = AFPUT@
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = COBECHO@
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = AREA1
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = AREA2
//INCLUDE-MOD  LIB = LIB.UTM.PRELINK                                -
//          ,ELE = AREA3
//MODIFY-SYMBOL-ATTR   PUBLIC = YES
//END-SUB-LLM
//REMARK ------------------------------------------------------------
//REMARK ------------------------------------------------------------
//SET-EXTERN-RESOLUTION  SYMBOL-TYPE=REFERENCE , RESOLUTION=STD
//MODIFY-MAP-DEFAULT  PROGRAM-MAP = PAR( DEFINITION = ALL            -
//                                  ,INVERT         = ALL            -
//                                  ,REFERENC       = ALL)           -
//                               ,UNRESOLVED     = YES               -
//                               ,SORTED-PRO     = YES               -
//                               ,DUPLICATE      = YES               -
//                               ,OUTPUT= LIST.LINK.EXAMPLE-LLM
/REMARK ------------------------------------------------------------
//SAVE-LLM   LIB = LIB.LOAD-MODULE.STARTUP                          -
//              ,ELEM               = *INTERNAL                     -
//              ,FOR-BS2000-VERSION  = *FROM-OSD-V4(...)
//REMARK ------------------------------------------------------------
//END
```

As the example shows, it is sufficient to link all shareable modules that do not have the PUBLIC attribute in a sub-LLM in the linkage editor run and then set the PUBLIC attribute for the entire sub-LLM using the MODIFY-SYMBOL-ATTRIBUTES statement. In the list LIST.LINK.EXAMPLE-LLM - section *PROGRAM MAP* - you can then see which slice contains the individual objects.

3.2.4 Linking runtime systems

Many runtime systems have shareable parts. Together with BLS, this means that load times can be reduced and performance increased. Depending on the runtime system, you can:

- Load shareable parts as a subsystem. This is the best option, but is not possible for all runtime systems.
- Statically link shareable parts and load them in a common memory pool.
- Link the necessary parts of the runtime system to an LLM and then load them dynamically.
- Link the runtime system statically to the start LLM.

When linking a UTM application with SYSLNK.CRTE.PARTIAL-BIND, ensure that the CRTE version on the system on which the application is linked is not later than the CRTE version on the system(s) on which the application will run.

When linking the start LLM, the C and SPL runtime system and the ILCS must be incorporated for UTM (ILCS is available in the CRTE libraries). This is achieved by specifying the two following binder statements in this sequence:

```
//RESOLVE-BY-AUTOLINK LIB=$userid.SYSLNK.CRTE[.PARTIAL-BIND]
```

```
//RESOLVE-BY-AUTOLINK LIB=$userid.SYSLNK.UTM.070.SPLRTS
```

If you want to link your runtime systems statically, these two statements for C, COBOL, and SPL objects are sufficient. If you require additional runtime systems you must – unless otherwise specified in the respective manual – insert the RESOLVE-BY-AUTOLINK statements for the necessary runtime systems between the RESOLVE statements for CRTE and UTM.

It is better to use the partial bind linking technique when linking the runtime systems. The advantage of linking with SYSLNK.CRTE.PARTIAL-BIND is that the link and load times are reduced and the linked program occupies less disk space. If linking is carried out using the partial bind library, references to the runtime system are satisfied by means of connection modules. The modules from the runtime system required at runtime are dynamically loaded when the application is run.

If you require SYSLNK.CRTE or SYSLNK.CRTE.PARTIAL-BIND a second time - in addition to linking the start LLM - you must use the respective CRTE library again. An example of this would be linking the local task part of the COBOL runtime system; see example on "Linking the start LLM".

The possibilities for shareable loading are explained in the following sections. Information on which runtime routines are shareable and which are non-shareable can be found in the description of the respective runtime system.

Please note that a load module containing runtime system modules must never be exchanged while the application program is executing, as this may result in errors that are difficult to diagnose.

3.2.4.1 Shareable runtime system parts as subsystem

If your UTM application is running with the common runtime system environment CRTE (Common RunTime Environment), it is possible to load the shareable parts of CRTE via DSSM. Other runtime systems also offer this option.

The BS2000 system administrator is responsible for loading the subsystem; for further information see the CRTE manual.

As far as possible, you should use this procedure because it saves system resources. In particular, the runtime system for C and COBOL, which are contained in CRTE, must be loaded as a CRTE subsystem because they are also required by KDCROOT.

If the CRTE runtime system is preloaded, you can use this by using the SYSLNK.CRTE.PARTIAL-BIND library during linkage. The connection to the CRTE subsystem is then established by a connection module incorporated when linking the start LLM. If the CRTE runtime system cannot be accessed, the connection module dynamically loads the necessary parts from the CRTE library into the local task area.

3.2.4.2 Shareable runtime system parts in a common memory pool

If modules use a runtime system that cannot be loaded as a subsystem and is also not incorporated in the start LLM, you should pre-link all shareable modules of the runtime system for one programming language to form LLMs and have openUTM load them into a common memory pool.

The shareable parts of the runtime system must then be generated as follows:

```
LOAD-MODULE share-rts-part,LOAD-MODE=(POOL,..., NO-PRIVATE-SLICE),...
```

You must also generate the non-shareable part of the runtime system (also pre-linked to form an LLM):

```
LOAD-MODULE nonshare-rts-part,LOAD-MODE=STARTUP
```

UTM generation is simplified if you link the shareable and non-shareable parts to an LLM. This load module must be generated with:

```
LOAD-MODULE rts,LOAD-MODE=(POOL,..., STARTUP),...
```

When generating, please ensure that this LLM is specified before the other modules required by the runtime system, as otherwise there will be unresolved external references.Please note that the shareable parts of the C, SPL and FOR1 runtime systems must not be loaded in a common memory pool managed by openUTM.

3.2.4.3 Linking runtime systems to an LLM

For all runtime systems that do not have a PARTIAL-BIND library, you must determine which modules of the runtime system are required for the entire application. This is done as follows:

1. All application program units and all event exits of the desired language, as well as all shareable modules of the runtime system loaded as a subsystem or in a common memory pool, must be linked together to a sub-LLM.
2. The ILCS (contained in CRTE) must likewise be incorporated in the sub-LLM; must be the highest available version.
3. To resolve all external references to the runtime system, you must enter the RESOLVE-BY-AUTOLINK statement to the library of the runtime system of the language: RESOLVE-BY-AUTOLINK LIBRARY=*rts-library*.
4. Using the REMOVE-MODULES statement, you must then remove the sub-LLM (see 1. and 2.) from the LLM.
5. The load module must be generated in openUTM using the following statement:

```
LOAD-MODULE RTS-PRIVATE-KERNEL   -
            ,VERSION=001         -
            ,LIB=private-rts-lib -
            ,LOAD-MODE=STARTUP
```

3.2.5 Linking the start LLM

When linking the start LLM, a distinction is made between the following options:

- Linking the ROOT table module statically in the start LLM, or
- Loading the ROOT table module dynamically at the start of the application program. In this case, the name of the library containing the ROOT table module must be specified in the start parameters for TABLIB, and the name of the ROOT table module (PLAM element name) must be specified for ROOTNAME (see section “Start parameters for openUTM”). First, the KDCRTMN module must be linked statically.

The second procedure is more beneficial, because you do not need to relink the application if changes are made in the UTM generation. For the same reason, you should if possible save all program units in load modules that are linked dynamically.

Please note the following points for both procedures:

- The runtime systems for C and SPL, together with the ILCS, are required by the KDCROOT modules, which is why the non-shareable parts of the runtime system must always be linked statically.
- After linking, the start LLM must only have unresolved external references to runtime system modules which are either loaded as a subsystem before the application start or loaded dynamically at the application start.
- You should avoid saving more than one start LLM in the same library, because BLS attempts to resolve open external references from this library at the start. I.e. in addition to load modules loaded dynamically, the library must only contain a start LLM loaded with the START-EXECUTABLE-PROGRAM command. You can also deactivate the AUTOLINK function using parameters of START-EXECUTABLE-PROGRAM.
- The RESOLVE statement to the library SYSLNK.CRTE.PARTIAL-BIND orSYSLNK.CRTE must always be specified as the first RESOLVE statement for a runtime system and then for the UTM library. The library you use in the start LLM to link the necessary modules of the CRTE must also be specified when linking all load modules of the application. The library SYSLNK.CRTE.PARTIAL-BIND offers performance advantages and should therefore be used in preference.

Example

The following BINDER run contains all the statements required to link a start LLM. It is assumed that the application is running with the shared runtime environment CRTE and was installed using IMON.

The string *vvv* stands for the openUTM version (e.g 070 for V7.0).

```
/START-BINDER
//START-LLM-CREATION INTERNAL-NAME=start-llm                          -
//                  ,INTERNAL-VERSION=start-llm-version
//REMARK  +-----------------------------------------------------------+
//BEGIN-SUB-LLM-STATEMENTS SUB-LLM-NAME=ROOT-TAB-LLM ---------------------- 1
//INCLUDE-MODULES LIBRARY=tablib ,ELEMENT=root-module
//END-SUB-LLM-STATEMENTS
//REMARK  +-----------------------------------------------------------+
//INC-MOD ELEM=KDCRTMN ,LIB=<userid1>.SYSLNK.UTM.vvv ---------------------- 2
//REMARK  +-----------------------------------------------------------+
//BEGIN-SUB-LLM-STATEMENTS SUB-LLM-NAME=LM-SHARED-RTS --------------------- 3
//INCLUDE-MODULES LIBRARY=$userid2.SYSLNK.CRTE.PARTIAL-BIND, ELEMENT=ITCMADPT
//INCLUDE-MODULES oncall-load-module
//INCLUDE-MODULES startup-load-module
//INCLUDE-MODULES pool-load-module
//END-SUB-LLM-STATEMENTS
//REMARK  +-----------------------------------------------------------+
//RESOLVE-BY-AUTOLINK LIBRARY=$userid1.SYSLNK.UTM.vvv --------------------- 4
//RESOLVE-BY-AUTOLINK LIBRARY=user-lib
//REMARK  +-----------------------------------------------------------+
//RESOLVE-BY-AUTOLINK LIBRARY=$userid2.SYSLNK.CRTE.PARTIAL-BIND ----------- 5
//RESOLVE-BY-AUTOLINK LIBRARY=other-rts-lib
//RESOLVE-BY-AUTOLINK LIBRARY=$userid1.SYSLNK.UTM.vvv.SPLRTS
//REMARK  +-----------------------------------------------------------+
//SHOW-MAP ..., UNRESOLVED-LIST=SORTED, ... ------------------------------ 6
//REMARK  +-----------------------------------------------------------+
//REMOVE-MODULES NAME=*ALL,PATH-NAME=.ROOT-TAB-LLM ----------------------- 7
//REMOVE-MODULES NAME=*ALL,PATH-NAME=.LM-SHARED-RTS ---------------------- 8
//REMARK  +-----------------------------------------------------------+
//SET-EXTERN-RESOLUTION SYMBOL-TYP=REFERENCES,RESOLUTION=STD
//SAVE-LLM LIBRARY=start-library,ELEMENT=*INTERNAL-NAME(           -
//                                VERSION=*INTERNAL-VERSION)     -
//               ,FOR-BS2000-VERSIONs=*FROM-OSD-V4(...)
//END
```

1. First of all, the ROOT table module is always linked in a separate sub-LLM. This module can be removed before saving the start LLM (see 7) if the ROOT table module, as in this example, is not linked statically but is to be loaded dynamically at the start of the application program.

   Using the external references of the ROOT table module, the static program units from *user-lib* are linked (see 4). However, these should be as few as possible because these modules cannot be exchanged.

2. Next, the KDCRTMN ROOT system module is linked. This must only be linked explicitly if a start LLM is to be created without a ROOT table module.

3. In this case, an adapter module (ITCMADPT) is linked into your own sub-LLM. This ensures that the necessary runtime modules are dynamically loaded.

   In addition, all existing load modules loaded dynamically are linked in this sub-LLM, so that the necessary runtime system modules are linked to the start LLM by the RESOLVE-BY-AUTOLINK statement. This means that they are already available in the dynamic loading process and need not be loaded dynamically with the autolink function, as this reduces the load performance.

   If you use the COBOL subsystem, the adapter module (ITCMADPT) can be omitted from the SUB-LLM mentioned above.

4. You must always specify the UTM library as the first library in a RESOLVE-BY-AUTOLINK statement. This can then be followed by RESOLVE-BY-AUTOLINK statements to user libraries.

5. You must always specify the CRTE library as the first library of a runtime system in a RESOLVE-BY-AUTOLINK statement. After the CRTE library, you should enter the RESOLVEs to any libraries of other runtime systems, before the library with the SPL runtime system required by openUTM. Further information can be found in the manual for the corresponding runtime system.
6. You can use the SHOW-MAP statement to list the unresolved external references, for example. The list, which is created at this point in the BINDER run, contains all open external references that are yet to be resolved. At the end of the BINDER run, you automatically receive another list of the unresolved external references, though this also contains all unresolved external references arising from the removal of the sub-LLM.
7. With a REMOVE-MODULES statement you remove the ROOT table module from the start LLM.
8. The adapter module (ITCMADPT) that has been linked in (if necessary) and the shareable modules of the runtime system for COBOL and the load modules generated with ONCALL, STARTUP or POOL are also removed from the start LLM.

The start LLM linked in this way must be loaded with the following command:

```
/START-EXECUTABLE-PROGRAM FROM-FILE=*LIBRARY-ELEMENT   -
                    (LIBRARY=start-library             -
                    ,ELEMENT-OR-SYMBOL=start-llm)      -
           ,DBL-PARAMETERS=*PARAMETERS(                -
           ,LOADING=*PARAMETERS(                       -
                    PROGRAM-MODE = *ANY                -
                    ,LOAD-INFORMATION = *REFERENCES))  -
           ,ERROR-PROCESSING=*PARAMETERS(              -
                    UNRESOLVED-EXTRNS=*DELAY           -
                    ,ERROR-EXIT = *NONE)
```

3.3 Information for applications with ILCS program units

- Program transitions (CALLs) between ‘ILCS’ program units or subroutines and ‘non-ILCS’ subroutines are prohibited.
- ILCS runtime system modules (prefix IT0....) must only be contained once in the loaded application program. In other words, you must ensure that the ILCS modules are not linked repeatedly when linking by RESOLVE from the RTS libraries of the compilers, and that they are not loaded repeatedly when modules are loaded dynamically.
- The highest available version of ILCS must always be loaded. The CRTE library always contains an ILCS.
- Please refer to the user guide or service information such as the Release Notice and readme files of the compiler for information on whether the respective compiler version or the runtime system supports ILCS and if so which version. Information on the ILCS capability of compilers and runtime systems can be found in the Release Notice.

  You will find a detailed list of the possible combinations of compiler options and RTS on "Compiler versions, runtime systems, KDCDEF options".
- The CRTE manual contains examples of linking ILCS programs.

4 Files required for operation

Before you start a UTM application, you must always ensure that the following files exist, as they are essential for the operation of the UTM application:

- the KDCFILE
- The system files SYSOUT and SYSLST are always present, but should be assigned to real, process-specific files.
- the system log file SYSLOG
- the user log file(s) USLOG (optional)
- all program and object module libraries from which modules should be loaded dynamically at startup and during operation of the application

KDCFILE, USLOG and SYSLOG (if used without a link name) have the prefix *filebase* (=base name of the UTM application). You must specify *filebase* in the start parameters. The name *filebase* is defined when creating KDCFILE with the generation tool KDCDEF, see openUTM manual “Generating Applications”, control statement MAX...,KDCFILE=*filebase*.

4.1 System files SYSOUT and SYSLST

openUTM logs messages to the system files SYSOUT and/or SYSLST i.e. openUTM writes all UTM messages with the UTM message destination SYSOUT or SYSLST to these files (see the openUTM manual

“Messages, Debugging and Diagnostics on BS2000 Systems” for information on message destinations). These system files should therefore be redirected to separate process-specific files.

You can switch these system files automatically during live operation. After you have switched the files, the old SYSOUT and SYSLST files can be evaluated and, if necessary, deleted in order to reduce the amount of disk space occupied.

After the BS2000 commands SET-SYSOUT-READ-MARK and SET-SYSLST-READ-MARK you can also access the system files that are still open (see “BS2000 OSD/BC commands” manual).

Switching system files

The system files can be switched over during live operation either by administration or at definable periodic intervals. The system files for all tasks of a UTM application are always switched over together, but the precise time at which this is done for individual tasks may be delayed when the system is under load.

- To switch the files over by administration
  - use the command KDCAPPL SYSPROT=NEW
  - use the *sysprot_switch* in the *kc_diag_and_account_par_str* data structure of the programming interface (see the openUTM manual “Administering Applications”)
  - use WinAdmin/WebAdmin

  The system files are switched over as soon as possible after the request has been made.
- To switch over the system files using a time interval, specify the start parameter SYSPROT when starting UTM application (see section “Start parameters for openUTM”). You can specify a time interval in days. The files are always switched over at midnight.

  If an error occurs when switching the files over, an error message is issued and time-controlled switching is deactivated.

Names of the switched files

When the UTM application is started, the system files are set up using the names specified either by the system or the user. Files are generated using the following name formats as of the first time that the files are switched over manually or automatically:

SYSOUT: `<prefix>.O.YYMMDD.HHMMSS.TSN`
SYSLST: `<prefix>.L.YYMMDD.HHMMSS.TSN`

The maximum total length of the new path name for a file that has been switched over is 54 characters and is made up as follows:

```
[:catid:][$userid.]filename-prefix.datei-suffix

<---6---><---10---><------17-----><----21---->
```

<prefix>

The prefix is made up of

catid, userid

Catalog ID and user ID under which the UTM application was started.
catid and *userid* can be omitted.

filename-prefix

The prefix you specified for the start parameter SYSPROT when starting the UTM application (see section “Start parameters for openUTM”).

The file name prefix can be a maximum of 17 characters in length.

Default value for <prefix>:
Name of the application specified in MAX APPLINAME during KDCDEF generation.

YYMMDD.HHMMSS

Date and time of the switchover

TSN

TSN of the task

4.2 System log file SYSLOG

openUTM logs events from the application run in the system log file SYSLOG (**SYS**TEM **LOG**GING), i.e. openUTM writes all UTM messages with the UTM message destination SYSLOG to this file (see the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems” for information on message destinations).

The system log file SYSLOG can be used for actively monitoring the application run or for subsequent checking. The SYSLOG file provides important information, particularly for diagnostic purposes.

You must provide a SYSLOG file for each UTM application. If you have not assigned a SYSLOG file to a UTM application before the start, the application start will be terminated.

The system log file SYSLOG can be created as:

- a simple SYSLOG file
- a file generation group SYSLOG-FGG

A SYSLOG-FGG has the following advantages over a simple SYSLOG file:

- You can switch to the next file generation during live operation (switchable SYSLOG file). You can administer the SYSLOG with the KDCSLOG administration command, for example. See the openUTM manual “Administering Applications” for more information. If all tasks of the application have closed the old SYSLOG file, then this file generation is available to you for your use.
- You can set automatic size monitoring for the SYSLOG. This means that you can generate or specify via the administration a threshold value for the size of the individual file generations of the SYSLOG-FGG. When this threshold is reached, openUTM automatically switches to the next file generation of the FGG. Size monitoring can be enabled and disabled while the application is running.

Messages from openUTM

openUTM outputs the following messages regarding the SYSLOG:

- Message K136 at the start of the application:

  ```
  K136 (First) SYSLOG file is <filename>
  ```

- Message K138 at the end of the application:

  ```
  K138 SYSLOG file <filename> closed
  ```

- Message K137 after switching to another file generation:

  ```
  K137 SYSLOG switched to file <filename>
  ```

4.2.1 SYSLOG as a simple file

If the SYSLOG file is to be maintained as a simple file, the SYSLOG file can be made known to openUTM in two ways:

- Before the start of the application, you assign the link name SYSLOG to the file (SET-FILE-LINK command). When the application starts, openUTM opens this file. It remains open for the entire application run for all tasks of the application.
- You can also create a file called *filebase*.SLOG before the start. The file must have the same catalog ID (CATID) as the KDCFILE *filebase*.KDCA (see openUTM manual “Generating Applications”, statement MAX...,CATID and KDCFILE). openUTM uses the file with the name *filebase*.SLOG as the SYSLOG file for each subsequent start, provided no file or file generation with the link name SYSLOG exists when the application starts. If a file or file generation with the link name SYSLOG does exist, openUTM always logs the SYSLOG UTM messages in this file.

  Following the end of the application run, you should save the contents of the *filebase*.SLOG file. With the next application start, the contents of this file are overwritten by openUTM.

> **CAUTION!**
>
> If you want to maintain the SYSLOG as a simple file, then you many **not** activate size monitoring for the SYSLOG. If you switch on size monitoring in the UTM generation with MAX..,SYSLOG-SIZE=*size* (*size* > 0), then openUTM aborts the start of the application with the start error code 58.

4.2.2 File generation group SYSLOG-FGG

When the SYSLOG file is maintained as an FGG, then openUTM opens a file generation of the SYSLOG-FGG as the SYSLOG file when the application starts. All tasks of the application write UTM messages with the destination SYSLOG to this file generation first.

If the SYSLOG is created as a file generation group, you need only create the file generation group (FGG) before the start. You do not have to create the individual file generations because openUTM does not open a SYSLOG file generation before checking that it already exists. If the file generation exists, it is simply opened. If the file generation does not yet exist, openUTM creates the file generation automatically with the properties of the file generation group.

If you want to assign particular values for primary and secondary allocation to the file generations, then you must create a file generation of the FGG with the desired values. openUTM then creates the next file generations with the same values.

Making the SYSLOG-FGG known

You have the following opportunities to make the SYSLOG-FGG known to openUTM:

- Using the link name “SYSLOG”

  At the start of the application, you create an FGG with any name for the SYSLOG. Then assign the link name SYSLOG to one of the FGG file generations using /SET-FILE-LINK. The link name must be assigned before each application start.

  In this case, the file generation group containing the file generation with the link name SYSLOG is taken as the SYSLOG-FGG. openUTM first logs data in the file generation with the link name SYSLOG (first SYSLOG file generation).

  If you generated automatic size monitoring for your application, openUTM switches to the next file generation of this FGG as soon as the size of the first SYSLOG file generation has reached the threshold value for size monitoring. If this file generation does not yet exist, openUTM creates it automatically.

  If the application was generated without automatic size monitoring, openUTM continues logging data in the file generation with the link name SYSLOG until you switch to another file generation of the FGG using the administration command KDCSLOG, or until you activate automatic size monitoring (e.g. with the administration command KDCSLOG). This file generation is likewise created automatically by openUTM when switching files.

- Creating an FGG with the name *filebase*.SLOG before the first start of the application

  This file generation group must have the same base name (including CATID and USERID) as the KDCFILE (KDCA file) and must be set up under the BS2000 user ID under which the UTM processes run.

  openUTM only uses the FGG with the name *filebase*.SLOG as the SYSLOG-FGG if no file or file generation assigned to the link name SYSLOG exists under the user ID of the application when the application starts. If a file or file generation with the link name SYSLOG exists, openUTM always logs the SYSLOG UTM messages in this file.

  If *filebase*.SLOG is a file generation group, then the defined base of the FGG determines which file generation is taken as the first SYSLOG file.

  Basis outside of the valid range

  If the base lies outside the valid range (e.g. BASE-NUM=0), then openUTM creates the file generation with the generation number LAST-GEN+1 at the start of the application. This file generation is then the first SYSLOG file.

  The next generation of the file generation last written in the previous application run is used by openUTM as the first SYSLOG file the next time the application starts. In other words, if data is written in the generations up to the *n*-th file generation in the last application run, then logging starts with the (*n*+1)-th file generation with the next application start.

  Basis within the valid range

  If the base lies within the valid range between the first and last file generation (the output of the SHOW-FILE-ATTRIBUTES command on the FGG shows FIRST-GEN <= BASE-NUM <= LAST-GEN), then the base generation is taken as the first SYSLOG file.

  openUTM does not modify the base number set by you. In other words, if the base lies outside the valid range, at the next application start openUTM begins logging again in the same file generation as with the previous start unless you modify the base setting beforehand.

4.2.2.1 Creating the SYSLOG-FGG

If you want to work with a SYSLOG-FGG, you must create this before the start of the application.

Creating SYSLOG as *filebase*.SLOG

In the simplest case you work with the FGG *filebase*.SLOG, whereby the base lies outside the valid range. This FGG need only be created once before the first application start. For each subsequent application start, openUTM automatically continues with the next generation of the file generation last written, provided you do not move the base into the valid range. You can create the FGG using the following BS2000 command:

```
/CREATE-FILE-GROUP                                  -
/   GROUP-NAME = filebase.SLOG                      -
/  ,GENERATION-PARAMETER = *GENERATION-PARAMETER(   -
/      MAXIMUM = n                                  -
/    [,VOLUME = volume                              -
/     ,DEVICE-TYPE = device                         -
/     ,OVERFLOW-OPTION = overflow ] )
```

Meaning of parameters:

MAXIMUM=n

Maximum number of file generations that can be cataloged simultaneously in the FGG.

VOLUME=volume, DEVICE=device

Volume identifier and device type of the disk on which the FGG is to be created. The FGG can be created on PUBLIC or PRIVATE DISK.

OVERFLOW-OPTION=overflow

Specifies what should happen if the maximum permitted number of file generations (MAXIMUM) is exceeded. You can use this operand to control whether only the last *n* (MAXIMUM) file generations are to be retained from the SYSLOG-FGG of your application, or whether all file generations written by openUTM are to be retained. You can specify CYCLIC-REPLACE, REUSE-VOLUME or KEEP-GENERATION.

(See also section “Identifier overflow protection” and section “Retaining SYSLOG generations”.)

KEEP-GENERATION

All file generations are retained, even if the number specified in MAXIMUM is exceeded.

CYCLIC-REPLACE and REUSE-VOLUME

Specify that the oldest file generation of the FGG is deleted before the new one is created.

If you specify CYCLIC-REPLACE or REUSE-VOLUME here, the value you select for the MAXIMUM (number of file generations) should not be too small. After switching files, the “old” file generation, which is no longer being written by openUTM, can be kept open for a longer period by some tasks (if these are using a user program unit for a very long time). If you have permitted *n* file generations and if a task keeps the file generation *i* open, it is not yet possible to switch to file generation *i*+*n*. The BS2000 system reports a DMS error for this file generation. Automatic size monitoring is suspended until the administrator of the application successfully switches the SYSLOG file using the KDCSLOG command.

With OVERFLOW-OPTION=KEEP-GENERATION, however, it is also possible to switch files in this case.

Using the command you create an FGG whose base lies at 0, i.e. outside the valid range. openUTM automatically creates all file generations. With the first application start, openUTM creates the file generation with the generation number 1 and uses this as the first SYSLOG file.

Creating SYSLOG with link names

If you want to work with the link name SYSLOG, then you can create the SYSLOG using the same command as for a SYSLOG with *filebase*.SLOG (see above). For GROUP-NAME, you can specify any name *fgg-name*. Before each application start, you must then use the following BS2000 command to assign the link name SYSLOG to the file generation which should be opened by openUTM as the first SYSLOG:

```
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=fgg-name(*gen)
```

gen is the generation number of the file generation which openUTM should open as the first SYSLOG file after the application start.

4.2.2.2 Creating a file generation

Before opening a file generation, openUTM checks whether the respective file generation already exists. If the file generation does not exist, then openUTM creates it. You can also create the file generation yourself to define different values for PRIMARY-ALLOCATION and SECONDARY-ALLOCATION.

Creation of the file generation by openUTM

Depending on whether the FGG was created on PUBLIC or PRIVATE DISK, openUTM issues one of the following commands internally:

- FGG on PUBLIC DISK

```
/CREATE-FILE-GENERATION                          -
/   GENERATION-NAME = fgg-name(*gen)             -
/  ,SUPPORT = *PUBLIC-DISK(                      -
/      SPACE = RELATIVE(                         -
/         PRIMARY-ALLOCATION = prim-alloc        -
/        ,SECONDARY-ALLOCATION = sec-alloc ) )
```

- FGG on PRIVATE DISK

```
/CREATE-FILE-GENERATION                          -
/   GENERATION-NAME = fgg-name(*gen)             -
/  ,SUPPORT = *PRIVATE-DISK(                     -
/      VOLUME = volume                           -
/     ,DEVICE-TYPE = device                      -
/     ,SPACE = RELATIVE(                         -
/         PRIMARY-ALLOCATION = prim-alloc        -
/        ,SECONDARY-ALLOCATION = sec-alloc ) )
```

Meaning of parameters:

fgg-name Name of the file generation group transferred to openUTM

gen Generation now to be opened

PUBLIC-DISK or PRIVATE-DISK

Specifies whether the file generation is to be created on public disk or private disk. openUTM takes this information from the catalog entry of the transferred SYSLOG-FGG.

VOLUME=volume, DEVICE=device

Volume identifier and device type of the disk on which the FGG is to be created.

PRIMARY-ALLOCATION=prim-alloc, SECONDARY-ALLOCATION=sec-alloc

Size of the initial allocation of storage space or the size of the storage space expansion.

If openUTM creates all generations of the FGG automatically, openUTM sets 192 PAM pages for both parameters. If openUTM has already opened an existing file generation of this FGG (one you created yourself) before creating the file generation, the values in the existing file generation for primary and secondary allocation are transferred by openUTM when creating subsequent file generations.

Creating the file generation yourself

If you want to define the values for primary and secondary allocation yourself for all file generations of the SYSLOG-FGG, you must create at least one file generation with the desired storage space specifications and transfer this file generation to openUTM as the first SYSLOG file. All subsequent file generations are then automatically created by openUTM with the specified values for primary and secondary allocation.

You can create the file generation using the following BS2000 command:

```
/CREATE-FILE-GENERATION                      -
/   GENERATION-NAME = filebase.SLOG(*1)       -
/  ,SUPPORT = *PUBLIC-DISC(                  -
/      SPACE = RELATIVE(                     -
/         PRIMARY-ALLOCATION = prim-alloc     -
/        ,SECONDARY-ALLOCATION = sec-alloc ) )
```

If the FGG is located on a private disk, you must also create the file generation on private disk.

Transferring the file generation to openUTM

You then transfer this file generation to openUTM as the first SYSLOG file. This can be done using the link name SYSLOG or by specifying the FGG base (see "File generation group SYSLOG-FGG").

If you work with the second method, then you must make sure that openUTM has not changed the base. If the FGG base is within the valid range of the FGG and you do not set the base to another generation before the next application start, then openUTM also begins with the same (base) file generation as the first SYSLOG file for the next application start.

The information from the previous application run may then be lost. Moreover, regardless of what you specified for OVERFLOW-OPTION when creating the FGG, only the *n* most recent file generations are retained (*n*=MAXIMUM in CREATE-FILE-GROUP). Please note section “Retaining SYSLOG generations”.

To ensure that openUTM begins with the next generation as the first SYSLOG after one application ends and another starts, you should carry out the following steps before each start.

1. Set the base to the last FGG generation written (LAST-GEN):

```
/MODIFY-FILE-GROUP-ATTRIBUTES                                -
/   GROUP-NAME = filebase.SLOG                               -
/  ,GENERATION-PARAMETER = *GENERATION-PARAMETER(            -
/      BASE-NUMBER = RELATIVE-TO-LAST-GEN( 0 ) )
```

2. Create the next file generation:

```
/CREATE-FILE-GENERATION                                      -
/   GENERATION-NAME = filebase.SLOG(+1)                      -
/   ......
```

3. Set the base to the file generation just created:

```
/MODIFY-FILE-GROUP-ATTRIBUTES                                -
/   GROUP-NAME = filebase.SLOG                               -
/  ,GENERATION-PARAMETER = *GENERATION-PARAMETER(            -
/      BASE-NUMBER = *RELATIVE-TO-LAST-GEN( 0 ) )
```

   You can only enter this command sequence once for each application start, not for each started UTM task.

4.2.2.3 UserId overflow protection

Proceed as follows to check the SYSLOG-FGG disk space:

1. Switch on automatic size monitoring either in the UTM generation using MAX...,SYSLOG-SIZE=*size* or by administration (e.g. using the administration command KDCSLOG SIZE= *size)*. In both cases, a value > 0 must be specified for *size*.
2. Create the SYSLOG-FGG using the following command:

```
/CREATE-FILE-GROUP                                    -
/   GROUP-NAME = filebase.SLOG                        -
/  ,GENERATION-PARAMETER = *GENERATION-PARAMETER(     -
/      MAXIMUM = n                                    -
/     ,OVERFLOW-OPTION = CYCLIC-REPLACE  or REUSE-VOLUME )
```

The file generations are cyclically overwritten so that a maximum of *n* generations are cataloged in the FGG. In addition, it may happen that another (older) file generation occupies storage space internally in the BS2000. With size monitoring, each generation has a maximum of *size* UTM pages.

This means that the maximum space occupied by the SYSLOG-FGG is less than or equal to: (*n*+1) * *size* * (size of a UTM page).

4.2.2.4 Retaining SYSLOG generations

If you want to retain all file generations of the SYSLOG-FGG, you must create the SYSLOG-FGG as follows:

```
/CREATE-FILE-GROUP                                          -
/   GROUP-NAME = fgg-name                                   -
/  ,GENERATION-PARAMETER = *GENERATION-PARAMETER(           -
/      MAXIMUM = n                                          -
/     ,OVERFLOW-OPTION = *KEEP-GENERATION )
```

In this case the base is set to 0, i.e. it lies outside the valid range between the first and last file generation. The BS2000 then retains all generations of the FGG, regardless of the value specified in the MAXIMUM parameter (maximum number of generations).

> **CAUTION!**
>
> If, on the other hand, you set the base of the FGG within the valid range, i.e. between the first and last generation, all generations outside the valid range (generation numbers not within LAST-GEN - MAXIMUM and LAST-GEN) are lost (without warning).

If the base lies within the valid range of the FGG and if you nonetheless want to keep as many generations as possible, you must select a sufficiently high value for *n* (preferably 9999). In this case, you can also specify REUSE-VOLUME or CYCLIC-REPLACE for the OVERFLOW-OPTION operand.

4.2.2.5 Automatic size monitoring

Automatic size monitoring can only be used for FGGs. If you create the SYSLOG file as a simple file and generate automatic size monitoring, then openUTM terminates the start of the application with start error code 58.

Automatic size monitoring can be set in two ways:

- in the UTM generation using the KDCDEF statement MAX ...,SYSLOG-SIZE=*size*
- while the application is running, using the administration command KDCSLOG [SWITCH,]SIZE=*size* or on the administration program interface with the operation code KC_SYSLOG and subopcode KC_CHANGE_SIZE (see the openUTM manual “Administering Applications”)

In both cases, you must set a value > 0 for *size*.

When size monitoring is switched on, openUTM does not write any UTM message to the SYSLOG file before checking whether writing this UTM message would exceed the agreed maximum size of the file generation (*size* * size of a UTM page). If this is the case, an attempt is made to switch to the next file generation. If successful, openUTM outputs UTM message K137.

If the attempt to switch generations results in an error, openUTM continues to work with the old file generation in which data was logged before the switching attempt was made. openUTM writes UTM message K139 to SYSOUT and to the administrator console. In addition, UTM message K043 is output for all DMS errors. This contains a DMS error code indicating the reason for the switching error.

To ensure that openUTM does not unsuccessfully attempt to switch to the next file generation for each subsequent UTM message with the destination SYSLOG, automatic size monitoring is deactivated after this type of switching error.

After the administrator has found and eliminated the cause of the switching error, automatic size monitoring can be reactivated using the KDCSLOG SWITCH command, for example. When KDCSLOG SWITCH is issued, openUTM is forced to begin a new switching attempt. If this attempt runs without errors, the previously deactivated size monitoring function is automatically reactivated.

openUTM guarantees that no more UTM messages are written to the old file generation following the switch. However, this does not mean that the old file generation is freely available immediately. First, all tasks of the application must close this file generation. This may take longer if tasks are using user program units for a very long time. When a file generation is closed by the last task, openUTM outputs UTM message K138.

The output of the administration command KDCSLOG INFO indicates which file generations are closed (LOWEST-OPEN-GEN).

openUTM does not change the FGG base set by the user at the start of the application. This means that file generations are not lost unintentionally even if the FGG option OVERFLOW-OPTION = KEEP-GENERATION is set.

4.2.3 Behavior in the event of write errors

If an error occurs in the attempt to write a UTM message in the SYSLOG, openUTM outputs UTM message K043, which contains a DMS error code. This error code indicates the reason for the error.

The subsequent behavior of openUTM depends on whether the SYSLOG is maintained as a simple file or as an FGG.

- The SYSLOG is maintained as a simple file:

  After UTM message K043 is output, the application is terminated with reason SLOG09.

- The SYSLOG is maintained as an FGG:

  When an error occurs, openUTM attempts to switch to the next file generation. openUTM also switches generations if size monitoring is deactivated or not generated. openUTM does not switch generations if size monitoring is suspended as a result of a previous switching error.

  If the switching attempt fails, the application is terminated with reason SLOG09.

  If openUTM can successfully switch to the next file generation, openUTM makes another attempt to write the UTM message in the SYSLOG. If an error occurs in this attempt, the application is terminated with SLOG09. If no errors occur, the application continues running and openUTM logs the UTM messages in the new SYSLOG file generation.

4.3 User log file

The user log file contains the records created by the application program with LPUT calls. The user log file must be created as a file generation group.

openUTM does not write the user log records directly into the log file, rather saves them first of all in the page pool of the KDCFILE. If the number of UTM pages in the page pool generated with MAX...,LPUTBUF=number are used by LPUT records, openUTM copies the records to the user log file. The records are copied outside the user transaction. If the application is terminated normally, openUTM likewise copies the records to the user log file.

The number of UTM pages specified in LPUTBUF=*number* must be taken into account when generating the size of the page pool with MAX...,PGPOOL=*number*.

The MAX...,LPUTLTH=*length* statement affects the block length of the user log file. It is calculated by openUTM and can be greater than the standard block of 2KB.

openUTM can only copy LPUT records to the user log file if this file is created and can be accessed by openUTM.

Note that the user log file is overwritten from the start following a KDCDEF or KDCUPD run; otherwise, data is added to the end of the file. For this reason, you should evaluate the log records before a KDCDEF or KDCUPD run.

4.3.1 Creating the user log file

The file generation group for the user log file must have the file name *filebase*.USLA. Here, *filebase* is the base name of the KDCFILE as generated in MAX..., KDCFILE=*filebase*.

The file generation group must be created before the first application start. The following BS2000 commands must thus be issued:

- CREATE-FILE-GROUP command to create the catalog entry. You must specify:
  - the name of the file generation group
  - the maximum permitted number of generations
  - the procedure when the maximum number of generations is reached (OVER-FLOW-OPTION=CYCLIC-REPLACE is the default)
  - the access authorization, if necessary
- CREATE-FILE-GENERATION command to create at least one generation.
- MODIFY-FILE-GROUP-ATTRIBUTES command to define the reference generation for relative numbering.

Example: Creating a file generation group for a user log file

```
/CREATE-FILE-GROUP GROUP-NAME=filebase.USLA                      -
/                 ,GENERATION-PARAMETER=GENERATION-PARAMETER(    -
/                  MAXIMUM=3,OVERFLOW-OPTION=REUSE-VOLUME)
/CREATE-FILE-GENERATION GENERATION-NAME=filebase.USLA(+1)
/MODIFY-FILE-GROUP-ATTRIBUTES GROUP-NAME=filebase.USLA           -
/                 ,GENERATION-PARAMETER=GENERATION-PARAMETER(    -
/                  BASE-NUMBER=RELATIVE-TO-LAST-GEN(NUMBER=0))
```

If the OVERFLOW-OPTION=REUSE-VOLUME parameter is set, a new generation is created on the same volume as the deleted generation.

If the user log records (LPUT calls) can be > 6 KB, you must specify the SPACE operand in the CREATE-FILE-GENERATION commands for the primary and secondary allocation. The values of the SPACE operand must be large enough that at least one LPUT record fits in the disk area for the secondary allocation. The primary allocation must be at least twice as big as the secondary allocation.

If the file generation group is to be created on a private disk, you must also note the following:

- Each individual file generation of the group must be created on the private disk using a CREATE-FILE-GENERATION command before the start of the application.
- You must specify OVERFLOW=REUSE-VOLUME when creating the file generation.
- If the base of the file generations is set to the last member of the file generation group following creation, openUTM begins with the last file generation and then switches to the next generation. If the first generation is specified as the base, openUTM switches to the last generation when the first KDCLOG command is issued (see Switching to the next file generation).

These restrictions do not apply to file generations on PUBLIC DISK.

Example: Creating a file generation group on private disk

```
/CREATE-FILE-GROUP GROUP-NAME=filebase.USLA                        -
/                  ,GENERATION-PARAMETER=GENERATION-PARAMETER(    -
/                   MAXIMUM=3,OVERFLOW-OPTION=REUSE-VOLUME        -
/                  ,VOLUME=B004H,DEVICE-TYPE=D3465)
/CREATE-FILE-GENERATION GENERATION-NAME=filebase.USLA(*1)         -
/                  ,SUPPORT=PRIVATE-DISK(VOLUME=B004H,DEVICE-TYPE=D3435)
/CREATE-FILE-GENERATION GENERATION-NAME=filebase.USLA(*2)         -
/                  ,SUPPORT=PRIVATE-DISK(VOLUME=B004H,DEVICE-TYPE=D3435)
/CREATE-FILE-GENERATION GENERATION-NAME=filebase.USLA(*3)         -
/                  ,SUPPORT=PRIVATE-DISK(VOLUME=B004H,DEVICE-TYPE=D3435)
/MODIFY-FILE-GROUP-ATTRIBUTES GROUP-NAME=filebase.USLA            -
/                  ,GENERATION-PARAMETER=GENERATION-PARAMETER(    -
/                   BASE-NUMBER=RELATIVE-TO-LAST-GEN(NUMBER=0))
```

4.3.2 Double user log file

If you are working with double user log files (MAX...,USLOG=DOUBLE), a second file generation group called *filebase*.USLB must be created in the same way as the first file generation group *filebase*.USLA. The file generation group *filebase*.USLB must be located in the same catalog as the parts of the KDCFILE with the suffix B.

4.3.3 Switching to the next file generation

The KDCLOG command and the KC_USLOG opcode on the KDCADMI program interface can be used by the administrator to switch to the next file generation. Each time the KDCLOG command or a corresponding KDCADMI call is issued, openUTM switches to the next file generation.

At the first start of the UTM application, openUTM writes the user log records in the generation of the group that was specified as the base at the time of creation. openUTM issues a MODIFY-FILE-GROUP-ATTRIBUTES command to the next file generation and sets the base to the next most recent file generation with each additional switch initiated by the administrator.

You can also switch to the next generation as follows after a UTM application terminates normally:

```
/CREATE-FILE-GENERATION GENERATION-NAME=filebase.USLA(+1)
/MODIFY-FILE-GROUP-ATTRIBUTES GROUP-NAME=filebase.USLA,                    -
/                              GENERATION-PARAMETER=GENERATION-PARAMETER(   -
/                              BASE-NUMBER=RELATIVE-TO-LAST-GEN(NUMBER=0))
```

These commands for switching to the next file generation can also be transferred to the start procedure of your UTM application.

If the administrator, for example, wants to switch to another user log file with KDCLOG, the file generation group must be created under the same BS2000 user ID under which the UTM processes are running.

4.3.4 Response to write errors

If a DMS (**D**ata **M**anagement **S**ystem) error occurs while writing LPUT records in the user log file, then openUTM outputs message K043, which contains a DMS error code. You can determine the reason for the error with this error code.
At the same time, every additional LPUT call in the program unit is rejected with the KDCS return code 40Z (internal return code K903).

The administrator of the application can then correct, restore or recreate the user log file or its generations.

The administrator must issue the KDCLOG administration command or a KDCADMI call with opcode KC_USLOG so that openUTM can write LPUT records to the user log file again (see the openUTM manual “Administering Applications”).

The file generation number is incremented. The LPUT records saved in the page pool of the KDCFILE are subsequently written to the log file(s). The lock for LPUT calls in the program units is released.

5 Starting a UTM application

A UTM application can be started as an ENTER process from the BS2000 console or from any terminal.

The CPU time etc. for the application is calculated under the BS2000 user ID used for this purpose.

All the necessary files must be cataloged either under this ID or under another ID on the same host with SHARE=YES.

The following files are required:

- library with a start LLM
- KDCFILE
- user log file (USLOG, optional)
- system log file (SYSLOG)
- user files of the application
- module library SYSLNK.UTM.070
- library with the connection module for the database system (optional)
- library with the connection module for the format handling system (optional)
- library with the formats (optional)

The ENTER or procedure file for starting the UTM application contains:

- CREATE-FILE- and SET-FILE-LINK commands for the system log file
- CREATE-FILE- and SET-FILE-LINK commands for user files (optional)
- START-EXECUTABLE-PROGRAM command for calling the application program
- Start parameters for UTM, FHS or the database system

When an ENTER file has been created, the application is started by calling the BS2000 command ENTER-JOB:

```
/ENTER-JOB FROM-FILE=enterfile[,JOB-CLASS=job-class]
          [,RESOURCES=PARAMETERS(CPU-LIMIT=tttt)]
```

In the case of a procedure file, use the BS2000 command ENTER-PROCEDURE to start the application:

`/ENTER-PROCEDURE FROM-FILE=enter-proc-file` (with the same parameters)

Recommendations for the selection of the parameters:

- You should set up a separate job class in which you set the most important parameters for the ENTER job of a UTM application.

  You can assign the job class then in the ENTER-PROC command or in the ENTER-JOB command (or concerning the parameters in an ENTER-JOB file in the SET-LOGON-PARAMETERS command).

- The operand CPU-LIMIT (CPU time limit) should be specified in the ENTER-PROC command or in the ENTER-JOB command if it is not specified in the SET-LOGON-PARAMETERS command in the ENTER-JOB file.

  The value should be set such that the jobs of a UTM application are not subject to any CPU time restriction. For this reason, you should set CPU-LIMIT=NO (or TIME=NTL) or define the job class accordingly.

  If the CPU time is limited, i.e. if CPU-LIMIT `not equal` NO has been set, and a CPU time runout occurs for a task of the application, then this can lead to the abnormal termination of the UTM application!

  Please note that an NTL authorization (**N**o **T**ime **L**imit) may be necessary in the user entry for the corresponding account number so that the Enter process can run without a CPU time limit.

  You can set CPU utilization limits for individual program units of a UTM application in the KDCDEF generation in the TIME operand of the TAC statement.

All the requirements for operating the application are fulfilled when the UTM application starts, i.e. the areas and tables are created, files are opened, connections are established, etc. These actions may result in error situations, which are identified by the start routine and which may lead to the termination of the application start or of a task. openUTM then outputs UTM message K078 or K049 to SYSOUT, which indicates the reason for the termination (see also the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”).

In the start procedure, no commands need be issued to check the consistency of the files opened by openUTM (REPAIR-DISK-FILES or CHECK-FILE-CONSISTENCY); openUTM automatically calls these commands when opening files that were not closed correctly.

The application program remains loaded until it is terminated either by the interception of the administrator or as a consequence of an error. The manner in which openUTM terminates the application program is important for the structure of the start procedure by the user.

The following situations can occur:

- On a normal termination using the administrative tools (e.g. via WinAdmin/WebAdmin or the administration command KDCSHUT NORMAL/WARN/GRACE) or after the application has terminated (Term Application abnormally), openUTM terminates the application program with TERM UNIT=STEP, i.e. all commands up to the next /SET-JOB-STEP or up to /EXIT-JOB or /LOGOFF are ignored in the ENTER procedure.
- When exchanging the application program with KDCAPPL PROG=NEW or following a program error which results in PEND ER, the application program should be loaded dynamically and restarted. openUTM then terminates the program with TERM UNIT=PRGR, i.e. the next command of the ENTER procedure is interpreted. This should be a /SKIP-COMMANDS command, which leads back to the /START-EXECUTABLE-PROGRAM command for starting the application program.

The processing in the start procedure following the /SET-JOB-STEP command can be rendered dependent on whether the UTM application was terminated normally or abnormally:

If the application is terminated abnormally (Term Application), openUTM creates an entry with LINK-NAME=KDCTRMAP in the task file table (TFT). In the start procedure, you can query (with /SHOW-FILE-LINK) whether such an entry exists in the TFT and control the subsequent procedure accordingly. When a UTM dump is written, the LINK name KDCDUMP is assigned for this purpose. In the start procedures, the user can execute the UTM-Tool KDCDUMP immediately to analyze the dump, for example.

The monitoring with job variables offers another means of restarting openUTM after an abnormal termination (see section “Restarting after an abnormal application termination”).

The start procedure is described in the following sections. It is contained in a file named *enterfile* or *enter-proc-file* in this section.

With the start procedure you create an ENTER process. The user ID used must be specified in the ENTER-JOB or ENTER-PROC command or in the SET-LOGON-PARAMETERS command of the start procedure.

UTM processes that are started by ENTER-JOB or ENTER-PROC are batch processes. They are thus subject to the JOB-CLASS restrictions for batch tasks. Normally, however, you want to start UTM processes immediately. This can be achieved using the “JOB EXPRESS” function. In this case, you must specify START-IMMEDIATE=YES in the user entry of the BS2000 user ID under which the processes are to run.

In addition to the batch and dialog tasks, there is also the class of TP processes on BS2000 systems. This is handled as a priority by the operating system. The processes started with ENTER-JOB or ENTER-PROC are registered by openUTM as TP processes. However, these processes are only handled as a priority if TP processes are permitted under the respective BS2000 user ID or job class. If TP processes are prohibited, UTM processes run as batch processes.

All the processes of an application must be started under the same BS2000 user ID.

i

- A UTM application can also be started interactively. This should only be done for test purposes, see also openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.
- The start procedure can also be started using WinAdmin/WebAdmin. For this purpose, the openFT product must also be installed in addition to WinAdmin/WebAdmin (on BS2000 systems and on the WinAdmin or WebAdmin computer).

- For an example of a start procedure, refer to the section “Basic structure of an SDF start procedure”

5.1 Start parameters of the application

The main routine KDCROOT reads the start parameters from SYSDTA when the application is started.

The start parameters define the current runtime parameters of the application. This includes the number of tasks with which the application is to start or parameters for a database and/or formatting system configured, for example.

The start parameters must be specified in two blocks:

- 1st block: start parameters for openUTM
- 2nd block: start parameters for database system and format handling system

The sequence of start parameters is arbitrary within a block. Each block – including an empty block – is concluded with an END command. If no database system and no format handling system is generated, two END commands are specified in succession.

UTM start parameters in the second block are ignored. This also applies to start parameters of the database system and format handling system in the first block.

The start parameters can be entered in one or more lines. A prefix determines who the start parameters are for:

- Start parameters with the prefix “.UTM” or without a prefix are interpreted by openUTM itself.
- Start parameters with the prefix “.UDS”, “.LEASY”, “.RMXA”, “.DB”, or “.CIS“ are forwarded by openUTM to the connected database system for evaluation.

  Example

  If the application was generated with DATABASE...,TYPE=UDS, for example, “.UDS” must be specified as the prefix for the parameters to be forwarded to UDS.

  If the application was generated with DATABASE...,TYPE=XA, “.RMXA” must be specified as the prefix.
- Start parameters with the prefix „.FHS“ are forwarded by openUTM to the FHS format handling system, which was generated with the type “.FHS” in the FORMSYS statement.

5.1.1 Start parameters for openUTM

The syntax of the UTM start parameters is illustrated below:

```
[.UTM] START   FILEBASE=filebase  [ ,CATID=(catalog-A,catalog-B) ]

               [ ,ADMI-TRACE= { ON | OFF } ]
               [ ,ASYNTASKS=number ]
               [ ,BTRACE={{ ON | OFF } | ({ ON | OFF }, length )}]
               [, CPIC-TRACE = { TRACE | BUFFER | DUMP | ALL | OFF }]
               [ ,DB-CONNECT-TIME=sec ]

               [ ,DBKEY=db-key ]

               [ ,DUMP-CONTENT={ STANDARD | EXTENDED } ]
               [ ,DUMP-MESSAGE=(event-typ,event) ]

               [ ,DUMP-PREFIX=filename-prefix ]

               [ ,DUMP-USERID={ STANDARD | SYSUSER } ]
               [ ,ENTER-PROC-INPUT='enter-proc-file,[(par1=param1

                                  ,...,par<n>=param<n>),<enter-proc-pars>]']

               [ ,OTRACE={ ON | (SPI, INT, OSS, SERV, PROT) | OFF } ]

               [ ,PASSWORD=connection-password ]

               [ ,ROOTNAME=rootname ]
               [ ,STARTNAME={ enterfile |

                            'enterfile[,enteroperand][...]' } ]

               [ ,START-SSL-PROC=[start-ssl-procedure,<enter-proc-pars>]

               [ ,STXIT={ ON | OFF } ]
               [ ,STXIT-LOG={ ON | OFF } ]

               [ ,SYSPROT=(interval,filename-prefix) ]

               [ ,TABLIB=libname ]
               [ ,TASKS=number ]

               [ ,TASKS-IN-PGWT=number ]

               [ ,TESTMODE={ ON | OFF | FILE } ]
               [ ,TX-TRACE = { ERROR | INTERFACE | FULL | DEBUG | OFF }]

               [ ,UTM-MSG-DATE={ YES | NO } ]

               [ ,XATMI-TRACE = { ERROR | INTERFACE | FULL | DEBUG | OFF }]

[.UTM] END
```

With the syntax shown above, the parameters are specified in a line without a carriage return and are separated by commas.

The UTM start parameters in the START command can be specified over several lines. In this case, the START command must appear in each line before the operands.

Syntax check at the start of the application

- If openUTM detects a syntax error in the start parameters, it outputs message K038, sets the corresponding value of the start parameter to its default value (if any) and starts the application.
- The application **cannot** be started when there is a syntax error in the FILEBASE parameter because there is no default value for this parameter.

Meaning of the commands

START This command is used to specify the UTM start parameters required for a UTM application run. The application is started as soon as all start parameters have been entered.

END This command concludes the input of start parameters (openUTM, database system, and format handling system).

The start parameters must be specified in own blocks:

- first the openUTM start parameters, and then
- the start parameters for the connected database and formatting systems.

Each block is concluded with the END command.

Meaning of the operands

FILEBASE = filebase

Base name of the KDCFILE and the user log file. Here you must specify the name under which the KDCFILE is stored at startup time (with a maximum of 42 characters in length). If an invalid name is specified, the application start is aborted.

ADMI-TRACE=

Enable/disable the ADMI trace function (= trace function for the KDCADMI administration program interface), see also openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

For information on the names of the trace files, see “Trace files” (Start parameters for openUTM).

ON The ADMI trace function is enabled at the start of the application.

OFF The ADMI trace function remains disabled at the start of the application.

Default: OFF

ASYNTASKS = number

Maximum number of tasks that are to work for asynchronous services.

Default value: Number defined in MAX...,ASYNTASKS=*number*
Minimum value: 0
Maximum value: Number defined in MAX...,ASYNTASKS=*number*

BTRACE =

Enable/disable the BCAM trace function.

ON

The BCAM trace function is enabled at the start of the application.
All network specific events are recorded in the BCAM trace file. In order for the trace to be written, it is necessary to set up a trace file in the UTM start procedure for each task and use the SET-FILE-LINK command to assign it the link name KDCBTRC.

For further information on setting up the trace file and analyzing it using the utility program KDCBTRC and for a description of the trace, see the openUTM manual

“Messages, Debugging and Diagnostics on BS2000 Systems”

OFF

The BCAM trace function remains disabled at the start of the application.

Default: OFF

length

Specifies the maximum length of data recorded when the BCAM trace function is activated. If the data to be recorded is longer, the first *length*/2 characters and the last *length*/2 characters of the data are recorded. This length can only be specified in the start parameters.

Default: 256
Minimum value: 32
Maximum value: 32624

If you use the BCAM trace for the UPIC Capture function (see also section“Recording the UPIC conversation (UPIC Capture)”) then it is
advisable to use the maximum value.

CATID = (catalog-A,catalog-B)

With this parameter you can specify the catalog IDs to which the parts of the KDCFILE are assigned. If you are working with CATID, you must specify the base name without CATID in the FILEBASE parameter. In the case of dual-file operation, you can assign the files with suffix A to *catalog-A* and the files with suffix B to *catalog-B*. If you are working with dual-file operation and if you only specify *catalog-A*, both files will be assigned to the same CATID.

CPIC-TRACE =

Enable/disable the CPI-C trace function (= trace function for the X/Open interface CPI-C), see also openUTM manual “Creating Applications with X/Open Interfaces”.

For information on the names of the trace files, see “Trace files” (Start parameters for openUTM).

TRACE

The CPI-C trace function is enabled with the level TRACE at the start of the application. The content of the input and output parameters is output for each CPI-C function call. Only the first 16 bytes are output from the data buffers. The return codes of the KDCS calls to which the CPI-C calls are mapped are output.

BUFFER — The CPI-C trace function is enabled with the level BUFFER at the start of the application. This trace level includes the TRACE level. However, the data buffers are logged in their full length.

DUMP — The CPI-C trace function is enabled with the level DUMP at the start of the application. This trace level includes the TRACE level and also writes diagnostic information to the trace file.

ALL — The CPI-C trace function is enabled with the level ALL at the start of the application. This trace level includes the levels BUFFER, DUMP and TRACE.

OFF — The CPI-C trace function remains disabled at the start of the application.

Default: OFF

DB-CONNECT-TIME = sec

Maximum time in seconds the system waits to establish a connection to the database.

If no connection is established to the database during this wait time, message K078 is issued and the task is terminated.

Default: 0 (no timeout)Minimum value: 60Maximum value: 3600

DBKEY = db-key

Key comprising up to eight characters for accessing the database. Only valid for UDS databases!

db-key is used when calling the database system if no DBKEY is defined for the transaction code of the calling program unit.

Default value: C’UTM ’

DUMP-CONTENT =

Specifies whether openUTM dumps the global task storage areas in all dumps of a dump file generation, i.e. for all tasks, or only in the dump of the task that caused the application crash.

STANDARD (STD) — If openUTM creates a dump file generation, global task storage areas are only contained in the dump of the first process (initiator). This is normally sufficient for diagnostic purposes.

Default value: STANDARD

EXTENDED (EXT) — The global task storage areas are contained in all dumps of a DUMP file generation.

> This value should only be set if explicitly requested by the system service.

DUMP-MESSAGE = *(event-type, event)*

Event for which UTM creates a UTM dump when test mode is enabled. A dump is only created by the task in which the event occurred; the application is not terminated.

The following can be specified for *event-type, event*:

event-type=MSG,*event*=K*nnn* (K message)

The UTM dump is created when message K*nnn* occurs. A dump is only created once for message numbers K023, K043, K061, K062; thereafter, *event* is reset automatically.With all other messages, a dump is created each time the message number occurs until the value is reset by the administrator.

> The value of DUMP-MESSAGE can be reset by the administrator, e.g. using WinAdmin/WebAdmin or by issuing the command KDCDIAG DUMP-MESSAGE=*NONE.

event-type

=RCCC,*event*=rccc (compatible KDCS return code)

Specify a KDCS return code (KCRCCC, e.g. 40Z) for rccc. When this return code is returned for a KDCS call, the process in which the return code occurred generates a

UTM dump e.g. with dump code CC-40Z. The message dump for this event is then automatically deactivated.

event-type=RCDC,*event*=rcdc (internal KDCS return code)

Specify a KDCS return code (KCRCDC, e.g. KD10) for rcdc. When this return code is returned for a KDCS call, the process in which the return code occurred generates a UTM dump `e.g. with dump code DCKD10`. The message dump for this event is then automatically deactivated.

event-type=SIGN,*event*=sign (SIGN status code)

Specify a SIGNON status code (KCRSIGN1/2, e.g. U05) for sign, where KCRSIGN1 must have the value U, I, A or R. If this code is issued when a user signs on, the process in which the SIGNON status occurred generates a UTM dump e.g. with dump code SG-U05. This happens irrespective of whether a signon service has been generated in the application or not. The message dump for this event is then automatically deactivated.

Notes

For all KDCS return codes >=70Z and the associated incompatible KDCS return codes for which no PENDER dump is written (e.g. 70Z/K316), no dump is created either.

Up to three different events can be specified in the administration command KDCDIAG or by the corresponding function at WinAdmin/WebAdmin or KDCADMI using the parameters DUMP-MESSAGE1, DUMP-MESSAGE2 and DUMP-MESSAGE3. In contrast, only one event can be specified using the start parameter *DUMP-MESSAGE*. In addition, no message inserts can be specified for *event-type*=MSG in the start parameter. By contrast, up to three inserts can be specified as additional conditions in the KDCDIAG command.

The dump code depends on the event:

Event	Prefix	Example
K or P message	ME followed by the message number	`MEP012`
Primary KDCS return code	CC- followed by the return code	`CC-71Z`
Secondary KDCS return code	DC followed by the return code	`DCK303`
SIGN status	SG- followed by the status	`SG-U01`

DUMP-PREFIX

filename-prefix is the filename prefix for UTM dump files. It can be a maximum of 17 characters in length.

If you specify *filename-prefix*, openUTM writes the memory dumps to a file generation group (FGG) or to a normal BS2000 file (dump file) with the name

filename-prefix `.rrrrrr.ttttff`.

If you do not specify a *filename-prefix* or if the memory dump (without blanks) is written at a very early phase during startup of the application, openUTM writes the memory dumps to a file generation group or to a file with the name

`DUMP.UTM.rrrrrr.ttttff.aaaaaaaa`.

`rrrrrr`	Return code specifying the reason for termination. You will find detailed information in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.
`tttt`	TSN of the task that created the dump.
`ff`	If the dump file is not an FGG file: Sequence number for the dumps created by one process of an application. If the dump file is an FGG file: Hexadecimal value of the application start counter since KDCDEF generation.
`aaaaaaaa`	Name of the application to which the dump belongs.

DUMP-USERID =

Specifies the user ID under which the UTM dumps are to be stored.

STANDARD	The UTM dumps will be stored under the user ID that started the UTM task.
(STD)	Default: STANDARD
SYSUSER	The UTM dumps will be stored under the $SYSUSER user ID.
(SYS)	

ENTER-PROC-INPUT = *enter-proc-input*

Specification of the ENTER-PROC file and its parameters.
When this parameter is specified, the follow-up tasks of a UTM application are started by means of an ENTER-PROC command.

enter-proc-input may be a maximum of 255 characters in length.

This parameter can be specified more than once. The specification must be enclosed in single quotation marks. The parameters specified in multiple ENTER-PROC statements are assembled in the order in which they are specified. The maximum length of the aggregate specifications is 2000 characters. If the maximum total length is exceeded, a message is output and the start of the application is aborted.

The collected *enter-proc-input* parameters are issued by UTM by means of an ENTER-PROC command. They must consequently be specified in the correct syntax of the ENTER-PROC command.

Example:

```
.UTM START ENTER-PROC-INPUT='START-DYN-PROC,(PAR1=<par1>,PAR2=<par2>'
.UTM START ENTER-PROC-INPUT=',PAR3=<par3>),LOGGING=*NO'
```

This issues the following command:

```
ENTER-PROC START-DYN-PROC,(PAR1=<par1>, PAR2=<par2>,
                                  PAR3=<par3>),LOGGING=*NO
```

In the event of syntax errors in this command, the message K048 is output, and no follow-on task is started.
The simultaneous specification of the ENTER-PROC-INPUT and STARTNAME parameters is rejected with the message K039, and the start of the application is aborted.

OTRACE =

Enable/disable the OSS trace function.
The OSS trace is required for diagnostic purposes if problems arise with OSI TP connections of the application. See also the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems” and the OSS manual.

ON — Enables the OSS trace function on the start of the application.Trace records of types SPI, INT, OSS, SERV and PROT are logged. When the OSS trace function is switched on, each process of the application creates its own trace file.

(SPI,NT,OSS, SERV,PROT) — Enables the OSS trace function on the start of the application. Trace records of the specified type are logged. The trace records are specified in an arbitrary sequence.

SPI — The XAP-TP system programming interface is logged.

INT — The internal processes in the XAP-TP module are logged.

OSS — The OSS calls are logged.

SERV — The internal OSS trace records of type O_TR_SERV are logged.

PROT — The internal OSS trace records of type O_TR_PROT are logged.

OFF — The OSS trace function remains off when the application starts.

Default: OFF

PASSWORD = connection-password

A 4-character password that must be specified by the user to establish a logical terminal connection to the application (see section “Standard sign-on processfor terminals”).

Default value: No password (X’0000 0000’)

ROOTNAME = rootname

The parameter is mandatory if the ROOT table module is to be loaded dynamically. *rootname* is the PLAM element name of the ROOT table module in the library specified with TABLIB=.

STARTNAME =

> The simultaneous specification of the ENTER-PROC-INPUT and STARTNAME parameters is rejected with the message K039. The start of the application is aborted.

enterfile — Name of the file specified in the ENTER command.
The ENTER file must be cataloged under the user ID under which the application was started.
enterfile may be up to 54 bytes long.

enteroperand — Operand of the BS2000 command ENTER. openUTM starts the follow-up tasks with the operands specified. The notation and effect of the operands specified correspond to the operands of the ENTER command
The operands must be specified in the ISP format.
The character string between the quotation marks
(’ *enterfile* , *enteroperand* , . . . ’) can be up to 200 characters long.

START-SSL-PROC = enter-ssl-proc-input

This parameter is only required if you have generated a BCAMAPPL with SECURE (see chapter Starting a UTM application Starting the Application with TLS Connections)

Specifies your own START procedure for the SSL proxy and, if necessary, parameters for the ENTER PROCEDURE call, without the procedure parameters required to start the SSL proxy (these are automatically generated by UTM and specified when the procedure is called).

Length: 255, the parameter can be specified several times, specified in single quotation marks, they are put together in the order in which they are specified.

The maximum length of the collected data is 2000 characters.

You should start the job for SSL proxy like the UTM tasks without CPU time limitation. For this reason, you should set CPU LIMIT=NO(or TIME=NTL) or define the job class accordingly.

If the CPU time is limited, that is, CPU LIMIT is not set to NO, and a CPU time runout occurs for the job, the SSL proxy is terminated abnormally.

Please note that an NTL authorization (NoTimeLimit) in the user entry for the relevant billing number may be required for the execution of enter tasks without a CPU time limit.

Example:

.UTM START START-SSL-PROC='OWN.START.SSL,JOB-NAME=ownnam,JOB-CLASS=jcbname'

This sends the following command:

ENTER-PROC OWN.START.SSL,JOB-NAME=ownnam,JOB-CLASS=jcbnam, PROC-PAR=(par1,par2,par3..5) -> this part is appended by UTM

If syntax errors occur in this command, a K099 message appears and the application continues to start (but no SSL proxy is started).

STXIT =

Enable/disable STXIT routines.

ON — The UTM-STXIT is enabled on the start of the application.

Default value: ON

OFF — The UTM-STXIT remains disabled. This is only possible for UTM applications started interactively.

STXIT-LOG =

Enable/disable of extended STXIT logging in the event of problems involving STXIT processing. Several K099 messages are output at SYSOUT.

ON — The STXIT logging function is enabled at the start of the application.

OFF

The STXIT logging function remains diabled at the start of the application.

Default: OFF

SYSPROT =

Switch over system files SYSOUT and SYSLST

interval

Switchover interval in days

Default: 0 (no interval: the files are not switched over at a specified time interval, but only on request)
Maximum value: 364

filename-prefix

Prefix for the new file name of the system files that have been switched over. *filename-prefix* must not exceed 17 characters, see also "System files SYSOUT and SYSLST".

Default:
Name of the application specified in MAX APPLINAME during KDCDEF generation.

You will find a complete description of how to switch over the system log files in the section “System files SYSOUT and SYSLST”.

TABLIB = libname

The parameter is mandatory if the ROOT table module is to be loaded dynamically. The *libname* library must then contain the object of the ROOT table module.

TASKS = number

Number of BS2000 tasks that are to be started for the application.

Default value: Number defined in MAX...,TASKS=*number*

Minimum value: 1 *)
Maximum value: Number defined in MAX...,TASKS=*number*

If the application is to work with several processes, then the STARTNAME= start parameter or the ENTER-PROC-INPUT start parameter must also be specified.

*) If the application is generated with Program Wait (i.e. if either a TAC class or a TAC is generated with PGWT=YES) then a value of at least 2 must be specified for the TASKS start parameter.

> **i** In addition to the number of tasks defined in TASKS, UTM starts further tasks for an application. These are known as UTM system tasks. The purpose of the UTM system task is to ensure that applications continue to be responsive even under high loads. The system tasks only process selected jobs which are characterized first and foremost by short runtimes. When an application is started, UTM starts up to three additional UTM system tasks for the application depending on the number of started tasks (TASKS= number).

TASKS-IN-PGWT = number

Maximum number of processes that can simultaneously execute program units with blocking calls (e.g. the KDCS call PGWT) are permitted (PGWT= operand in the TAC and TACCLASS KDCDEF statements).

Default value: Number defined in MAX ...,TASKS-IN-PGWT=*number* in the UTM generation.
Minimum value: 1 if MAX...,TASKS-IN-PGWT > 0; otherwise 0.
Maximum value: Number defined in MAX ...,TASKS-IN-PGWT=*number* in the UTM generation.

TESTMODE =

Enable/disable test mode.
See also the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

ON

Test mode is enabled when the application starts. In test mode, additional internal UTM plausibility checks are executed and trace information is logged in the internal trace area. Test mode should only be switched on to diagnose UTM errors on the recommendation of the system service.

OFF

Test mode is to remain disabled when the application starts. In particular situations, dumps are suppressed after PEND ER if TESTMODE=OFF.

Default value: OFF

FILE

Test mode is enabled when the application starts. In addition, the diagnostic data is written to a file each time the internal trace area overflows so as to avoid any loss of diagnostic data. The file name is made up of the base name *filebase* and the TSN of the respective task, i.e. the following file is created for each task for a UTM production application:

filebase.KTATRC.*tsn*

TX-TRACE =

Enable/disable the TX trace function (= trace function for the X/Open interface TX), see also openUTM manual “Creating Applications with X/Open Interfaces”.

For information on the names of the trace files, see “Trace files” (Start parameters for openUTM).

ERROR

The TX trace function is enabled with the level ERROR at the start of the application. Only errors are logged.

INTERFACE

The TX trace function is enabled with the level INTERFACE at the start of the application. The level INTERFACE includes the level ERROR, and TX calls are also logged.

FULL

The TX trace function is enabled with the level FULL at the start of the application. The FULL level includes the INTERFACE level. All KDCS calls to which the TX calls are mapped are also logged.

DEBUG

The TX trace function is enabled with the level DEBUG at the start of the application. The level DEBUG includes the level FULL, and diagnostic information is also logged.

OFF

The TX interface trace function remains disabled at the start of the application.

Default: OFF

UTM-MSG-DATE =

Specifies whether UTM messages are output to SYSOUT/SYSLST with or without date /time.

YES

All UTM messages to SYSOUT/SYSLST are prefixed with the date/time.

Default: YES

NO

The UTM messages do not include any date/time prefix.

XATMI-TRACE=

Enable/disable the XATMI trace function (= trace function for the X/Open interface XATMI), see also openUTM manual “Creating Applications with X/Open Interfaces”.

For information on the names of the trace files, see “Trace files” (Start parameters for openUTM).

ERROR

The XATMI trace function is enabled with the level ERROR at the start of the application. Only errors are logged.

INTERFACE	The XATMI trace function is enabled with the level INTERFACE at the start of the application. The level INTERFACE includes the level ERROR, and all XATMI calls are also logged.
FULL	The XATMI trace function is enabled with the level FULL at the start of the application. The FULL level includes the INTERFACE level. All KDCS calls to which the XATMI calls are mapped are also logged.
DEBUG	The XATMI trace function is enabled with the level DEBUG at the start of the application. The level DEBUG includes the level FULL, and diagnostic information is also logged.
OFF	The XATMI interface trace function remains disabled at the start of the application. Default: OFF

Trace files

By default, the trace records of the ADMI, CPI-C, TX, and XATMI trace function are written to one of the following files:

`KDC.TRC.` *trace-type*.*appliname*.*tsn*

trace-type

Identifies the trace type:

ADMI

ADMI trace (link name: KDCADMI)

CPIC

CPI-C trace (link name: KDCCPIC)

TX

TX trace (link name: KDCTX)

XATMI

XATMI trace (link name: KDCXATMI)

appliname

Name of the application

tsn

TSN of the UTM task (4 digits)

You can also assign your own trace files via link names.

5.1.2 Start parameters for the database system

The notation and meaning of these parameters are described in the User Guides for the respective database systems. Additional information can be found in section “Starting and stopping a UTM database application”.

5.1.3 Start parameters for the format handling system

The notation and meaning of these start parameters are described in the User Guide for the format handling system (FHS). openUTM only accepts start parameters for the format handling system if this was specified in the KDCDEF statement FORMSYS in the UTM generation.

5.2 Starting the application

The application program is started using the START-EXECUTABLE-PROGRAM command.

The following then applies:

- If shareable parts of the application or the runtime systems required by the application are to be loaded into system memory, then this must be initiated by the administrator before the start of the application program.
- The START-EXECUTABLE-PROGRAM command loads the statically linked part of the application program into working memory. All load modules of the application program that were generated with LOAD-MODE=STARTUP or LOAD-MODE=(POOL,...) are loaded at the start of the application as independent units.
- If individual load modules cannot be loaded, then the start of the application is generally resumed. If there is a branch to a program that could not be loaded while the application is running, then this leads to a BADTACS event service call or to a PEND ER. If the MSGTAC, SIGNON event services or the START, SHUT, INPUT and FORMAT event exits or the administration program unit or AREAs could not be loaded, then the start of the application is aborted with an error message.
- openUTM checks on the basis of the generation information whether the application program loaded in this task matches the program started in tasks started earlier when an additional task is started for the application and also after a PEND ER. If the data does not match, then openUTM aborts the start of the follow-up task with an error message.

Start commands for the application program

The application program is started with:

```
/START-EXECUTABLE-PROGRAM FROM-FILE = *LIB-ELEM -
/              (  LIB = llm-plamlib             -
/               , ELEM = start-llm              -
/              )
```

The application program *start-llm* must be made available as a type L element in a program library with name *llm-plamlib* .

If you have generated a load module with ALTERNATE-LIBRARIES=YES, you must assign a link name (BLSLIB*nn* with 00<=*nn*<=99) to the library of the respective runtime system before the application program starts. While the application is running, you can output a DBL list of the dynamic load processes with the BS2000 command

`/MODIFY-DBL-DEFAULT PRIORITY=*FORCED, SCOPE=*ALL(PROGRAM-MAP=...).`

Because this delays the start process and the program exchange, you should only use this when debugging or when an error occurs.

If parts of the application program are to be loaded later, then you must specify the following operands at the start of the application program:

```
/     ,DBL-PARAMETERS = *PARAMETERS(                                    -
/          ,LOADING    = *PARAMETERS(                                   -
/                          PROGRAM-MODE = *ANY                          -
/                         ,LOAD-INFORMATION = *REFERENCES)              -
/          ,RESOLUTION = *PARAMETERS(                                   -
/                          ALTERNATE-LIBRARIES = *BLSLIB##              -
/                         ,AUTOLINK = *ALTERNATE-LIBRARIES )            -
/          ,ERROR-PROCESSING=*PARAMETERS(                               -
/                          UNRESOLVED-EXTRNS=*DELAY                     -
/                         ,ERROR-EXIT = *NONE))
```

The following list shows the options available to you for influencing the autolink function:

- If you do not need the Autolink function during the start phase and you have unresolved external references in the start LLM, then you should suppress the function with AUTOLINK=*NO.
- AUTOLINK=*YES,ALTERNATE-LIBRARIES=*NO causes the library from the load call to be searched.
- AUTOLINK=*ALTERNATE-LIBRARIES,ALTERNATE-LIBRARIES=*TASKLIB/*BLSLIB## causes the TASKLIB and/or the BLS libraries to be searched.
- AUTOLINK=*YES,ALTERNATE-LIBRARIES=*TASKLIB/*BLSLIB## causes the library from the load call and the TASKLIB and/or BLS libraries to be searched.

Even if the start LLM does not contain any unresolved external references at the start, the DBL search in the shared code should be suppressed with the start operand SHARE-SCOPE=*NONE.

openUTM first loads all load modules that are to be maintained as shareable in common memory pools (LOAD-MODE=POOL) at the start of the application. First, the common memory pools are loaded that were generate with SCOPE=GLOBAL, and then those with SCOPE=GROUP are loaded, in each case in the order specified in the generation. Then openUTM loads all load modules that were generated with LOAD-MODE=STARTUP also in the order in which you specified them in the LOAD-MODULE statement.

Load modules that were generated with LOAD-MODE=ONCALL are only loaded once a program unit of this load module is called.

i
- Different UTM applications should be started under different BS2000 user IDs, if possible, to prevent errors that occur due to having the same module names in shareable parts. Modules that are used by several applications should therefore be loaded into global common memory pools or in nonprivileged subsystems.

  A sample start procedure is also supplied with openUTM, see section “Sample procedures”.

Starting an application with TLS connections

If a BCAMAPPL with T-PROT=(SOCKET,...,SECURE) has been generated for a UTM application, communication takes place via this transport system access point via the Transport Layer Security (TLS).

When an application generated in this way is started, openUTM starts an additional task in which a reverse proxy is executed, which is used to handle all TLS connections in the form of a TLS Termination Proxy. The task with the reverse proxy is started with the job name "SSLPROXY". When the application is terminated, the reverse proxy is also terminated.

The reverse proxy writes a SYSOUT file with the following pattern: <APPLNAME>.<JOBNAME>.<TSN>.YYYY-MM-DD-HH:MM:SS . This file is toggled at midnight.

The reverse proxy communicates with HTTPS clients via TLS connections and with the UTM application via a host-local socket connection. The reverse proxy must run on the same computer as the UTM application.

As a prerequisite for starting a reverse proxy for a UTM application, a job variable with the base name of KDCFILE must be cataloged. The product JV ("job variable") is required for this. This job variable is used for the orderly termination of the proxy process.

openUTM starts the task with the reverse proxy via an ENTER procedure that is stored with the name START-SSL-PROXY in the SYSLIB.UTM.070.SSL library. If required, this procedure file can be adapted to the needs of the user.

In this case the procedure should be fetched from the library and adapted accordingly.

When the application is started, the new start parameter "START-SSL-PROC" is specified with the name of this procedure and any desired sequence parameters. The required procedure parameters such as filebasename, listener ports, etc. are supplied by openUTM.

If no start parameter is specified, the procedure is started from the above library.

The reverse proxy task reads SSL configuration parameters from the SAM file.

<filebase>.SSL.CONF

Such a file with the name FILEBASE.SSL.CONF is delivered by openUTM as an example. This file is also located in the SYSLIB.UTM.070.SSL library. It must be used by the user for his application and copied to the ID in which the UTM application is started. Before starting the application, the user must adapt the entries in this file to his environment and requirements.

The following parameters can be configured in this file.

CACertificateFile

The *CACertificateFile* option specifies a file that contains the CA certificates in PEM format required for the authentication of the reverse proxy.

CipherSuite

The *CipherSuite* option specifies a preferred list of encryption procedures

If this option is not specified, a default preference list is used.

DSACertificateFile

The *DSACertificateFile* option specifies a file that contains the DSA-based X.509 server certificate in PEM format.

This file can also contain the private DSAServer key. As a rule, however, the certificate and key are stored in separate files. In this case, the key file is specified using the DSAKey-File option.

DSAKeyFile

The *DSAKeyFile* option specifies a file that contains the private DSA server key in PEM format.

If both the X.509 server certificate and the private server key are contained within the same file, the RSAKeyFile option need not be specified.

RSACertificateFile

The *RSACertificateFile* option specifies a file that contains the RSA-based X.509 server certificate in PEM format.

This file can also contain the private RSA server key. As a rule, however, the certificate and key are stored in separate files. In this case, the key file is specified using the RSAKeyFile option.

RSAKeyFile

The *RSAKeyFile* option specifies a file that contains the private RSA server key in PEM format.

If both the X.509 server certificate and the private server key are contained within the same file, the RSAKeyFile option need not be specified.

ECDSACertificateFile

The *ECDSACertificateFile* option specifies a file that contains the ECDSA-based X.509 server certificate in PEM format.

ECDSAKeyFile

The *ECDSAKeyFile* option specifies a file that contains the private ECDSA server key in PEM format.

Further details on the syntax and meaning of the options, as well as on generating the certificate and key files are described in the manual "interNet Services User V3.4B" in chapter "SSL", as well as in the manual "interNet Services Administrator V3.4B" in chapter "TELNET Configuration".

5.3 Cold start and warm start

These terms are explained below for openUTM:

- Cold start:
  Start following a normal termination of the UTM application or following a regeneration.
- Warm start:
  Start following an abnormal termination of the UTM application.

Cold start with openUTM

Before an application starts for the first time, you create the KDCFILE using the generation tool KDCDEF. After a regeneration of the KDCFILE or if the UTM application was terminated normally before, openUTM performs a cold start the next time the application is started. After the application has been started successfully, openUTM outputs the message:

```
K051 Successful cold start for application <appliname> under UTM V07.0A00 /
operating-system-type
```

Warm start with openUTM

If a UTM application has been abnormally terminated, openUTM performs a warm start the next time this application is started. During a warm start, openUTM brings the KDCFILE into a consistent state. After a successful start, openUTM outputs the message:

```
K050 Successful warm start for application <appliname> under UTM V07.0A00 /
operating-system-type
```

You should note that UTM-S and UTM-F differ in the scope of their restart functions. See also the openUTM manual “Concepts und Functions”.

5.4 Error messages at the application start

If the start of a UTM application or of a process is terminated due to an error, openUTM generally outputs messages K049 and/or K078. Message K078 can occur in several variants. The meanings of these messages and their contained return codes are described in detail in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

The program branches to the next /SET-JOB-STEP command or to /EXIT-JOB or to the /LOGOFF command in the procedure.

When using SDF-P, BEGIN-/END-BLOCK and IF-BLOCK-ERROR should be used, see section “Basic structure of an SDF start procedure”.

Start errors can occur at the start of every process.

5.5 Restarting after an abnormal application termination

openUTM offers the option of an automatic restart after an abnormal application termination.

> The product JV “Job Variables” is a prerequisite for the procedure outlined below.

If a job variable is cataloged with the base name of the KDCFILE (KDCA, including catid and userid), this job variable is used by openUTM.

The first position of the job variable can have the value R or T:

R: UTM application is running

T: UTM application is terminated

The second position of the job variable can have the value N or A:

N: Normal termination

A: Abnormal termination

These job variables can be used to monitor the execution of a UTM application using an ENTER job which restarts the application if the application terminates abnormally. If the first column of the job variable contains the value “T”, the value of the second job variable column determines whether this ENTER job is terminated (with ‘N’) or the UTM application is restarted (with ‘A’).

The ENTER job can be structured as follows:

```
/SET-LOGON-PARAMETERS
/.WAIT1 REMARK
/WAIT-EVENT UNTIL=*JV(CONDITION=((jobvariable-name,1,1)='T'), -
/           TIME-LIMIT=32767,TIMEOUT-LABEL=WAIT1)
/SKIP-COMMANDS TO-LABEL=END,IF=*JV(CONDITION=((jobvariable-name,2,1)='N'))
/ENTER-JOB enterfile,CPU-LIMIT=ttt,JOB-CLASS=job-class
/WAIT-EVENT UNTIL=*JV(CONDITION=((jobvariable-name,1,1)='R'))
/WAIT-EVENT UNTIL=*JV(CONDITION=*NONE,TIME-LIMIT=600,TIMEOUT-LABEL=WAIT2)
/.WAIT2 REMARK
/SKIP-COMMANDS TO-LABEL=END,IF=*JV(CONDITION=((jobvariable-name,1,2)='TA'))
/SKIP-COMMANDS TO-LABEL=WAIT1
/.END REMARK
/EXIT-JOB
```

Description of the ENTER job

- The first WAIT-EVENT command instructs the job to wait until the UTM application has terminated (‘T’ in the first field of the job variable). If the application terminated abnormally (‘A’ in the second field of the job variable), the ENTER job is initiated to restart the UTM application; with a normal termination, the monitoring job is also terminated.
- With the second WAIT-EVENT command, the job waits until the UTM application is running again.

- With the third WAIT-EVENT command, the job waits for ten minutes (600 sec) until a query is issued as to whether the UTM application has terminated abnormally again. If so, the monitoring job is terminated and the UTM application is not restarted. If not, the system branches to the first WAIT-EVENT and the monitoring job lies in wait again. This prevents the UTM application from continuously terminating and restarting in the event of a permanent error.

5.6 Basic structure of an SDF start procedure

The following example shows an SDF start procedure. As recommended when using SDF-P, BEGIN-BLOCK, END-BLOCK, and IF-BLOCK-ERROR are used.

```
/SET-PROCEDURE-OPTIONS                                -
/        CALLER               = ANY                   -
/       ,IMPLICIT-DECLARATION = YES                   -
/       ,LOGGING              = YES                   -
/       ,INTERRUPT-ALLOWED    = YES                   -
/       ,INPUT-FORMAT         = FREE                  -
/       ,DATA-ESC             = STD                   -
/       ,SYS-FILE             = STD                   -
/       ,DATA-ERROR           = YES                   -
/       ,JV-REPLACEMENT       = AFTER-BUILTIN-FUNCTION
/BEGIN-PARAMETER-DECLARATION
/&* ------------------------------------------------
/&* here you can declare your Procedure-Parameters
/&* ------------------------------------------------
/END-PARAMETER-DECLARATION
/&* ------------------------------------
/&* here is place for your variables
/&* ------------------------------------
/ SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=<applname>.SYSLOG
/ MODIFY-TEST-OPTION DUMP=YES
/ ASSIGN-SYSDTA TO-FILE=*SYSCMD
/ ASSIGN-SYSOUT TO-FILE=SYSOUT.&(TSN()).<applname>
/&* ---------------------------------------------------------------
/&*  Here follows the section with the SET-FILE-LINK commands if
/&*   at least one LOAD-MODULE is defined with
/&*   ALTERNATE-LIBRARIES = YES.
/&*  If you need RFA connections, or an AID connection (AID-FE)
/&*   for testing an UTM production application or other
/&*   connections please insert here the commands.
/&* ---------------------------------------------------------------
/REPEAT: BEGIN-BLOCK
/ SHOW-FILE-LINK LINK-NAME=KDCDUMP
/&* ---------------------------------------------------------------
/&*  If you want to do some actions after PEND ER, then you must
/&*   insert commands here. (E.g. prepare an UTM dump.)
/&* ---------------------------------------------------------------
/ REMOVE-FILE-LINK LINK-NAME=KDCDUMP
/ GOTO EXEC
/ IF-BLOCK-ERR; END-IF
/&* ------------------------------------------------
/&* here is place for actions after KDCAPPL PROG=NEW
/&* ------------------------------------------------
```

```
/EXEC:
/ MOD-DBL-DEFAULTS SCOPE=*CMD-CALLS(               -
/      LOAD=*PAR(LOAD-INF=*REF),                   -
/      RESOLUTION=*PAR(SHARE-SCOPE=*NONE),         -
/      ERROR-PROC=*PAR(UNRESOLVED-EXTRNS=*DELAY),  -
/      REPORT=*PAR(MES-CONTR=*WARN,                -
/                  PROG-MAP=*SYSLST) )
/ START-EXEC-PROG FROM-FILE=*LIBRARY-ELEMENT (     -
/                    LIBRARY  = <libname>          -
/                    , ELEMENT  = <elem>)
.UTM START FILEBASE  = <applname>
.UTM START ROOTNAME  = <rootname>
.UTM START TABLIB    = <root-lib>
/&* ------------------------------------------------
/&* in this section you can insert the parameter(s)
/&* for node-recovery  for example:
/&* .UTM START NODE-TO-RECOVER=<nodename>,RESET-PTC=Y/N
/&* ------------------------------------------------
.UTM START ENTER-PROC-INPUT='STRT-ENTER-PROC'
/&* ------------------------------------------------
/&* if your enter-procedure has parameters you have to
/&* change the last line with correct syntax
/&* if necessary duplicate the parameter-line
/&* ------------------------------------------------
.UTM START TASKS     = 3
.UTM START TASKS-IN-PGWT     = 0
.UTM START ASYNTASKS = 2
.UTM START TESTMODE  = OFF
.UTM START STXIT     = ON
.UTM START BTRACE    = OFF
.UTM START OTRACE    = OFF
.UTM START DB-CONNECT-TIME = 0
.UTM END
/&* ------------------------------------------------------------
/&*  Here follows the section with the data base and format system
/&*   parameters (if necessary).
/&* ------------------------------------------------------------
.FHS MAPLIB          = <maplib>
END
/ GOTO REPEAT
/ IF-BLOCK-ERR
/ SHOW-FILE-LINK LINK-NAME=KDCTRMAP
/&* ----------------------------------------------------------
/&*  If you want to do some actions after abnormal application
/&*   termination, then you can put some commands here.
/&* ----------------------------------------------------------
/ END-IF
/ END-BLOCK REPEAT
 / GOTO EXIT
/ IF-BLOCK-ERR; END-IF
/&* ----------------------------------------------------
/&*  Here you can insert commands to execute after normal
/&*   application termination.
/&* ----------------------------------------------------
/EXIT:
/ REMARK
/EXIT-PROC
```

6 Terminating a UTM application

A UTM application can

- be terminated normally via the administration or
- abnormally as a result of a bottleneck of operating resources, as a result of internal errors in openUTM, or via the administration.

openUTM terminates all processes with TERMJ, i.e. the system branches to the next /SET-JOB-STEP command or to the /EXIT-JOB- or /LOGOFF command in the start procedure.

In the start procedure, you can query whether the UTM application was terminated normally or abnormally (TERMAP). See also section “Start parameters for openUTM”. The following query options are available:

- job switch 3
- TFT entry
- job variable

When using SDF-P, BEGIN-/END-BLOCK and IF-BLOCK-ERROR should be used, see section “Basic structure of an SDF start procedure”.

6.1 Terminating a UTM application normally via administration

The UTM administrator terminates a UTM application normally by entering the following UTM administration command at a terminal, for example:

```
KDCSHUT GRACE,TIME=time
```

or

```
KDCSHUT WARN,TIME=time
```

or

```
KDCSHUT NORMAL
```

Applications that use distributed transaction processing should always be terminated with KDCSHUT GRACE or WARN because this allows open distributed transactions to end properly.

When the application is terminated, openUTM performs the following actions:

- All jobs still in the UTM queue are processed.
- The connections to all communication partners of the application are shut down.
- The connection to the database system is shut down.
- The KDCFILE, system log file, and user log file are brought to a consistent state and closed properly.
- All processes of the application are terminated.

You can use the corresponding WinAdmin/WebAdmin function or administration program interface function instead of the KDCSHUT command to terminate a UTM application normally.

If BCAM is terminated with BCEND, any UTM application loaded at this time will be terminated normally.

If the UTM application is being monitored by a job variable (see "Restarting after an abnormal application termination"), the first position of the job variable is set to “T” and the second to “N”.

> ¡ A UTM task can only be terminated with the /CANCEL-JOB command if it is running in the non-privileged mode (TU). A program unit in an endless loop can be terminated, for example, with this command.

6.2 Terminating a UTM application abnormally

A UTM application is terminated abnormally by any of the following events:

- internal UTM error
- error in the system environment, e.g. database system not available
- UTM administration command KDCSHUT KILL (or by the corresponding WinAdmin/WebAdmin or KDCADMI function)
- UTM generation error

The following actions are performed when a UTM application is terminated abnormally:

- All transactions currently being processed by the individual processes are aborted immediately.
- The connections to all communication partners of the application are shut down.
- The connection to the database system is shut down.
- A UTM-specific dump is created for each work process of the application. See also the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.
- All processes of the application are terminated and all files are closed. No attempt is made to bring the KDCFILE to a consistent state. This does not occur until the application is restarted.

If the UTM application was being monitored by a job variable (see "Restarting after an abnormal application termination"), the first column of the job variable is set to “T” and the second to “A”.

Following an abnormal termination of the application, you must first determine the cause of the crash. To find the cause, look for message K060 in the SYSLST log. This message contains the dump error code as an insert. This error code gives precise information regarding the cause of the abnormal termination. You can also find the cause for the dump as part of the name of the UTM dump file. The meanings of the dump error codes are described in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”. There are three possibilities:

- The dump error code indicates that a KDCDEF operand must be modified.In this case, the KDCFILE must be regenerated. If you want to retain the application data in the page pool, proceed as follows:
  - warm start with ASYNTASKS=0, TASKS=1
  - terminate the application normally with KDCSHUT NORMAL
  - new KDCDEF generation with the modified operand
  - transfer the application data from the old to the new KDCFILE using KDCUPD
  - start the application with the new, updated KDCFILE

- The dump error code cites the cause as:
  - DMS error
  - memory bottleneck
  - the database system is currently not available

  When the error has been rectified, you can restart the application, and openUTM executes a warm start automatically.
- Otherwise (if the dump error code is not described) a system error has occurred. In this case, produce diagnostic documentation and write a problem report to the system support personnel. Please refer to section “Diagnostic documentation for a problem report” for information on which documentation is required by system support.

  A warm start with the same KDCFILE is not always successful in this case. If a warm start cannot be performed, you must regenerate the KDCFILE using KDCDEF.

6.3 Diagnostic documentation for a problem report

If a UTM application terminates abnormally due to a system error, openUTM writes dumps for all tasks of the application (see the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”).

The following diagnostic documentation must be supplied when you write a problem report for the system support personnel:

- UTM dump files of all tasks
- SYSOUT logs of all tasks
- system log file SYSLOG
- KDCDEF control statements

The UTM dumps and SYSLOG file should not be edited.

In addition, you should retain the following documentation in the event of subsequent queries:

- KDCFILE and, if it is split over several files, also all pagepool and restart files
- the log created when the program was linked
- The protocol of the KDCDEF run

7 UTM database application

This chapter provides a comprehensive overview of how to implement databases under openUTM.

The following systems are connected via the interface used for UTM-DB collaboration (IUTMDB):

- UDS/SQL
- SESAM/SQL
- LEASY (the LEASY file system behaves like a database system with respect to openUTM)
- CIS

The following systems are connected via the XA interface:

- Oracle

In addition, openUTM can also work in conjunction with other database systems or be connected via the XA interface if they support the either interface IUTMDB interface or the XA interface.

Multi-DB operation

openUTM can work in coordination with up to three different types of database systems, i.e. a UTM transaction can contain calls to multiple database systems. Of the database systems that have requested update operations, only one may not support a “two-phase commit” protocol.

Coordinated interoperation between openUTM and the two DB systems "in *a single* multi-database application" is available for UDS/SQL and SESAM/SQL if the following interfaces are used in this case:

- COBOL or Call-DML is used for calls to UDS/SQL
- SQL for calls to SESAM/SQL

However, the UDS/SQL and SESAM/SQL systems cannot be combined in one application if SQL is used for calls to the UDS/SQL DB system.

Multi-DB operation with more than three database systems (maximal up to 8 DB systems) is possible on special release.

Further details on the concept of coordinated interoperation can be found in the openUTM manual “Concepts und Functions”. More information is available in the DB system manuals under the topic of UTM or openUTM, e.g. “UDS/SQL - Programming Applications” or “SESAM/SQL - Core Manual”.

7.1 Generating a UTM database connection

For coordinated interoperation, you must generate the UTM database connection in the KDCDEF statement DATABASE. The XA connection on BS2000 systems must also be generated with the DATABASE statement. Here you specify:

- the type of database system

  The type “DB” must be specified for database systems from other manufacturers.
- the entry name of the database
- the library from which the database connection module can be loaded

  You can specify a symbolic link name for the library. This causes a search to be made for the connection module in the IMON installation path for SESAM/SQL and UDS/SQL databases; the module is then loaded from the library. This has the advantage that the UTM application is independent of installation IDs and library names of the database system.
- Database access data (user name, password).

  These specifications are optional and are only permitted for Oracle databases. If you want to store the access data in the UTM generation then you must use placeholders in the open string for the user name and the password.

You can specify the DATABASE statement more than once and in any order. If you define several database systems of the same type, the entry names must be different. You can combine all the database systems that you can specify during UTM generation; one of the systems can also be an external system (DATABASE=DB).

More details can be found in the openUTM manual “Generating Applications” under the description of the DATABASE statement.

Creating and assembling the ROOT source

During the KDCDEF run, a ROOT source is created which contains the following macro calls, depending on the specifications for DATABASE:

- macro call KDCDB when working with one database system
- macro calls KDCDB*x* when working with several database systems; here, *x* is the first letter of the database type specified in DATABASE, e.g. KDCDBS for SESAM

The KDCDB macro or KDCDBx macros are supplied with the respective database systems. When assembling the ROOT source, the libraries containing these macros must be assigned explicitly.

More details on creating the root source can be found in the openUTM manual “Generating Applications” in the chapter “Generating application components”. The creation of load modules is described in the section “Generating load modules”.

7.2 Linking a UTM database application

For coordinated interoperation with a database, openUTM needs a database connection module. This module is supplied with the respective database system. The library containing this connection module is an optional specification in the KDCDEF statement DATABASE (see above).

The connection module can either be loaded dynamically or linked when the application is linked. If openUTM is to interoperate with two database systems, a specific connection module is required for each database system.

Loading the connection module dynamically

The reference to the connection module can be omitted when linking the application program. The advantage of this is that the application program need not be relinked when upgrading to a new database version (with a new connection module). In this case, the connection module is loaded dynamically when the application starts.

The following search algorithm applies here:

1. in the link context
2. in the user's shared code
3. in non-privileged subsystems (e.g. the DB system can be loaded as a subsystem)
4. in the shared code of the system address space
5. in the library that was specified at UTM generation in the DATABASE statement
6. in the alternative libraries you declared using the link name BLSLIB*nn* (00<=*nn*<=99) at application startup with a SET-FILE-LINK command

In particular, this means that the connection module is **not** loaded from the library specified in the DATABASE statement if the DB system was loaded entirely as a non-privileged subsystem, or if parts of the DB system were loaded as non-privileged subsystems and the connection module is contained in one of these parts.

> **CAUTION!**
>
> When loading the connection module of the database system, please note that it might only be operable in 24-bit mode. In this case, the statement MODIFY-SYMBOL-ATTRIBUTES ... ADDRESSING-MODE=24 must be inserted when linking the UTM application.

Linking the connection module

If the connection module is to be linked when the application is linked, you should use the INCLUDE-MODULES statement. The RESOLVE-BY-AUTOLINK statement can produce undesirable side effects if the libraries specified with RESOLVE-BY-AUTOLINK contain other modules in addition to the connection module.

7.3 Starting and stopping a UTM database application

A UTM database application can be started and stopped in the same way as a UTM application without a database, i.e. by starting and stopping the UTM application program.

Please note here that the DBH (DataBaseHandler) of the database is always started **before** the UTM application, as otherwise an openUTM start error might occur.

7.3.1 Start parameters for a UTM database application

To start a UTM-DB application, the start parameters for the database(s) must be specified in addition to the start parameters for openUTM, if this is required by the DB system. These start parameters are database-specific and are described in the manuals of the respective DB system.

The start parameters must be entered in accordance with the following schema, whereby the start parameters of openUTM always appear **before** the start parameters of the database:

`.UTM`	Start parameters for UTM, see section “Start parameters of the application”; you may have to specify the DBKEY parameter.
	`...`
`.`*db-typ1*	Start parameters for the database of type *db-typ1*, see the manual for the DB system *db-typ1*.
`.`*db-typ2*	Start parameters for the database of type *db-typ2* if a second DB system is used; see the manual for the DB system *db-typ2*.
`.`*db-typ3*	Start parameters for the database of type *db-typ3* if a third DB system is used; see the manual for the DB system *db-typ3*.
	...

For *db-typ1*, *db-typ2*, *db-typ3* you must specify the database type you generated in the DATABASE statement.

You can specify the following for UDS, for example:

```
.UDS DATABASE=UDSCONF
```

This defines the UTM application connection to the UDS configuration UDSCONF.

> ¡ With SESAM/SQL, the start parameters must be supplied in a configuration file, which is assigned to the tasks of the UTM-SESAM application under the link name SESCONF.

7.3.2 Start parameters for a UTM database application with XA support

openUTM also supports the XA functionality for connections to database systems on BS2000 systems. The functionality when using traditional database systems on a BS2000 system is not affected by this (e.g. UDS/SQL, SESAM/SQL or LEASY).

The start parameters have the following general format:

```
.RMXA RM="...",OS="openstring",CS="closestring"
```

openstring	The format of *openstring* is database-specific and is described in the documentation for the database system. There you will also find the meanings of the individual parameters in the open string. openUTM passes the strings from the start parameter file to the database system without checking them.

The start parameters for a UTM-Oracle application, for example, have the following format:

```
.RMXA RM="Oracle_XA",OS="openstring"
```

At connection setup openUTM automatically replaces the generic placeholders
*UTMUSER and *UTMPASS by the user ID and the password which were generated in the RMXA statement.

Additional parameters are required for failover support in the Oracle® Real Application Cluster; see "Start parameters for failover with Oracle® Real Application Clusters".

Specifying access data for the database

You can specify the access data for the database (user name, password) either in the start parameters or in the UTM generation, whereby the following applies:

- Specification in the start parameters takes priority over UTM generation. You can either specify both user name and password or just the password or just the user name. In all three cases, the message K238 is output when starting the UTM application.
- If you want to use the access data from UTM generation (see "Using the Oracle user name and Oracle password from the UTM generation"), you must specify the placeholders *UTMUSER (for the user name) and *UTMPASS (for the password) in the start parameters. When establishing a connection, openUTM automatically replaces the generic placeholders *UTMUSER and *UTMPASS with the user name and password generated in the DATABASE statement.

For security reasons, storing the access data in the UTM generation is recommended.

The access data can be changed later by administration.

Example for the connection to an Oracle XA database

Start parameters:

```
.RMXA RM="Oracle_XA",OS="Oracle_XA+Acc=P/*UTMUSER
/*UTMPASS+SqlNet=RACSERVICE1+SesTm=60+DbgFl=0"
```

Appropriate KDCDEF generation:

```
DATABASE TYPE=XA,ENTRY=XAOSWD,USERID='MaxTheSuperman',PASSWORD='PasswordOfMax'
```

7.3.2.1 Multiple instances

When a connection is established over the XA interface, the UTM application can operate with several Oracle instances (databases). To do this, you must specify a separate open string for each instance. Each open string must be specified in a separate line in the start parameter file. The different instances are specified in the open strings.

The general syntax for start parameters of Oracle XA instances is:

```
.RMXA[xa-inst-name1] RM="Oracle_XA",OS="Oracle_XA[+DB=db-name1] ..."
.RMXAxa-inst-name2 RM="Oracle_XA",OS="Oracle_XA+DB=db-name2 ..."
.RMXAxa-inst-name3 RM="Oracle_XA",OS="Oracle_XA+DB=db-name3 ..."
```

Here the value of *xa-inst-name1*, *xa-inst-name2*, etc. must always match the generation value of the XA-INST-NAME parameter of an DATABASE statement of the KDCDEF generation.

Please observe the following three rules to ensure that the uniqueness of the assignment of XA instances to the UTM generation information is guaranteed in all cases:

- Use at most one empty XA instance name and at most one empty Oracle DB name per UTM application.
- All XA instance names must be unique in the UTM application.
- All Oracle DB names must be unique in the UTM application.

Example

This example shows the connection of two Oracle XA databases to openUTM (BS2000).

Start parameters:

```
.RMXA RM="Oracle_XA",OS="Oracle_XA+Acc=P/*UTMUSER
/*UTMPASS+SqlNet=RACSRV1+SesTm=60+DbgFl=0"
.RMXAEVE RM="Oracle_XA",OS="Oracle_XA+DB=EVESDB+Acc=P/*UTMUSER
/*UTMPASS+SqlNet=RACSRV2+SesTm=60+DbgFl=0"
```

The first start parameter contains only the prefix ".RMXA", i.e. an empty XA instance name, and is therefore assigned to the first database generated in openUTM without an XA-INST-NAME parameter.

The second start parameter contains the prefix ".RMXAEVE", i.e. the second XA instance is assigned to the generated database with XA-INST-NAME=EVE, and the access data generated for this ("Eve", etc.) is used by this XA Instance.

KDCDEF generation:

```
DATABASE TYPE=XA,ENTRY=XAOSWD,USERID='MaxTheSuperman',PASSWORD='PasswordOfMax'
DATABASE TYPE=XA,ENTRY=XAOSWD,USERID='Eve',PASSWORD='PasswordOfEve',XA-INST-NAME=EVE
```

7.3.2.2 Using the Oracle user name and Oracle password from the UTM generation

For security reasons, the access authorization for an Oracle database should be defined via KDCDEF generation.

Please note the following:

- The Oracle user name for the connection to Oracle and the associated Oracle password must be generated in KDCDEF (KDCDEF statement DATABASE, USERID and PASSWORD operands).

  The Oracle password is stored as a hashcode in the UTM system tables (masked) and is therefore not present in clear text in the UTM dump.
- In the open string for the start parameter, specify the placeholder *UTMUSER in place of the Oracle user name and the placeholder *UTMPASS instead of the Oracle password. These placeholders are replaced in accordance with the following rules:
  - If the open string contains at least one of the placeholders *UTMUSER or *UTMPASS, then UTM replaces the placeholders with the values generated for the specific database system on an xa_open() call. I.e. in the open string, *UTMUSER is replaced by the generated Oracle user name and *UTMPASS by the generated Oracle password.

    For security reasons, the Oracle password is converted into clear text only immediately prior to use on an xa_open() call and is then deleted in the process memory immediately after the xa_open() call.
  - If the open string of the start parameter does not contain either *UTMUSER or *UTMPASS then it is passed unchanged to the xa_open() call.

Please note that processing is case-sensitive!

Examples

You want to use the Oracle user name and the Oracle password from the UTM generation:

```
OS="Oracle_XA+SqlNet=O11+ACC=P/*UTMUSER/*UTMPASS+DbgFl=15"
```

Behavior if the Oracle access data is not generated

- If the USERID and PASSWORD operands were not specified during UTM generation then you can specify the Oracle user name and the Oracle password directly in the start parameters.
- If you specify *UTMUSER or *UTMPASS in the start parameter even though the USERID and PASSWORD operands were not specified during UTM generation then UTM uses an empty Oracle user name or empty Oracle password. As a result, the attempt to establish a connection to the database will generally be unsuccessful.

7.3.2.3 Start parameters for failover with Oracle® Real Application Clusters

A UTM application communicates with Oracle® Real Application Clusters over the XA interface. If an XA call cannot be executed correctly by Oracle in the event of a failover, Oracle returns the value "XAER_RMFAIL".
In normal circumstances, i.e. when failover support is not activated, openUTM takes this message to mean that it is no longer possible to work with this database and aborts execution of the application.

In order to prevent execution from being aborted in these circumstances, you should also specify the value RAC=Y under the .RMXA parameters and control behavior in the event of a failover with the optional parameters RAC_retry and RAC_recover_down:

```
.RMXA RM="Oracle_XA",OS="openstring" ,RAC=Y[,RAC_retry=nnn]
                                        [,RAC_recover_down={Y|N}]
```

RAC=

Y Enables failover support when connecting the UTM application to Oracle® Real Application Clusters.

N disables failover support.

Default: N

RAC_retry=nnn

nnn specifies the number of times that openUTM attempts to reconnect to the database and execute a recovery job.

If the Commit job could not be executed for a transaction which has the state "Prepare-to-Commit" as a result of a failover, openUTM reconnects to the database and executes a recovery job. If the current XID is contained in the list of supplied XIDs, openUTM executes a Commit job for that XID, i.e. for the current transaction. If the XID is not
contained in the list, openUTM performs an *xa_close*. Then openUTM again tries to connect to the database and execute a recovery job.

Default: 1

RAC_recover_down=

Specifies the behavior of openUTM if the transaction could not be finally completed after the number of attempts specified by RAC_retry=, i.e. if the status of the transaction could not be set to "Commit".

N openUTM assumes that the transaction is no longer known to Oracle® Real Application Clusters. The transaction is assumed to have the status "Commit" and openUTM continues execution of the application.

Default: N

Y openUTM terminates execution of the application abnormally and thus forces a warm start in order to ensure that the data is consistent.

Behavior of openUTM in the event of failover

If you have enabled failover support, openUTM and the database system behave as follows:

- The application is not aborted if failover to a node of the Oracle® Real Application Cluster is possible.
- If the connection is lost between "Prepare" and "Commit" at the end of a transaction, a "Reconnect" with recovery is performed and if this is successful, the "Commit" operation is repeated over this new connection.
- If transactions are still open when the failover occurs, this can still lead to problems and corresponding error messages even if failover support is enabled (e.g. return codeORA-25402 - transaction must roll back). The reason for this is that Oracle® Real Application Clusters is unable to migrate any open transactions in the event of a failover. These transactions must be rolled back by the UTM application program, see also “Interrupted transactions”.

Any open multi-step transactions (i.e. following PEND KP) are rolled back by the database system in the event of a failover. openUTM has no influence over this.

The database system is automatically reconnected after the rollback. It is then possible to start new transactions.

- If the failover occurs during a warm start of the application or while the UTM process is being terminated, error processing is carried out as usual and no attempt is made to reconnect.
- The "prepared statements" database function can lead to errors in the event of a failover.
- Messages allow the progress of the reconnection to the database system to be monitored.
  - xa_close in the event of reconnection:

    In the &RMSTAT insert in message K202, the string "RAC closed" is output for the Oracle® Real Application Clusters instance in place of "closed".
  - xa_open in the event of reconnection:

    In the &XACALL insert of message K224, the string "RAC: xa_open" is output.

  Debug messages

  The debug messages contain an indication whether the message refers to an instance of Oracle® Real Application Clusters. How to get the XA-DEBUG information for the connection to the database, is described in section “Debug parameters”.

Interrupted transactions

Interrupted transactions can only be continued by the node that started the transaction. For this reason, all UTM processes must always be connected to the same node of the Oracle® Real Application Cluster. It is therefore simplest to proceed as follows:

- terminate the UTM application after failover of the Oracle® Real Application Cluster and before the failed node is restarted,
- restart the UTM application after the failed node has been restarted.

This ensures that all UTM processes are connected to the same node of the Oracle® Real Application Cluster and that all transactions of the application are processed by the restarted node of the Oracle® Real Application Cluster.

If it is not possible to terminate and restart the UTM application, i.e. if the nodes of the Oracle® Real Application Cluster are switched over while the UTM application is running, this can result in the following situation in which not all UTM processes are connected to the same node:

- One transaction is interrupted by the failover; at this time, the UTM process is still connected to the old node.
- After the process is restarted or after a PEND ER in the UTM application program, the interrupted transaction is continued by a different UTM process. This process is now connected to the new node.
- The database instance rejects the request to resume the interrupted transaction (xa-start with RESUME) and reports that the transaction is unknown.
- openUTM reconnects to the database instance. openUTM attempts to resume the transaction over the new connection (i.e. with the new node).
- The database system again rejects this request, since the database transaction was started on the old node of the Oracle® Real Application Cluster and cannot be continued on the new node.
- openUTM rolls back the global transaction and issues a K160 message; "NOTA" is output in the insert of the internal return code KCRCDC.

A situation such as this can be handled as described below using a MSGTAC program.

Control using a MSGTAC program

The MSGTAC event service is defined as an additional message destination for the K160 message. MSGTAC reacts to the message insert and initiates a restart over the administration programming interface (KC_CHANGE_APPLICATION). The application program is reloaded and then connects them to the new node.

This method minimizes the period of time for which the UTM processes are connected to different nodes. The number of transactions that are rolled back is limited to those that were started on the old node and could not be continued on the new node. The transactions that were started on the new node before the restart can be continued.

Oracle® connection

Connection to an Oracle® database is established using a "service". You can also set up "DTP services" in an Oracle® Real Application Clusters environment.

This offers the following options for live operation:

- automatic error detection
- automatic failover.
  If an instance fails, a new transaction is redirected to another instance of the service. No administrator intervention is required.
- load distribution as soon as the connection is established

Creating a DTP service (Oracle®)

Use the command "srvctl add service" to add a new service for the database and assign it to an instance of the database.

Example:

Two "DTP services" are to be created with the following options for the RAC database `dbracutm` with the instances `racutm1` and `racutm2`:

-d	Name of the database
-s	Name of the (DTP) service
-r	Name of the first instance
-a	Name of the second instance
-P	Failover method

```
"srvctl add service -d dbracutm -s racutmS12 -r racutm1
                                             -a racutm2
                                             -P BASIC"
```

and

```
"srvctl add service -d dbracutm -s racutmS21 -r racutm2
                                             -a racutm1
                                             -P BASIC"
```

The service `racutmS12` connects to the instance `racutm1` and to the instance `racutm2` in the event of a failover. In the same way, the service `racutmS21` connects to the instance `racutm2` and to the instance `racutm1` in the event of a failover.

Convert the services to "DTP services“ using SQLPLUS:

```
SQL> connect ....
SQL> execute dbms_service.modify_service
             ( service_name => 'racutmS12', dtp => true );
SQL> execute dbms_service.modify_service
             ( service_name => 'racutmS21', dtp => true );
SQL> exit
```

You can start, stop and administer the (DTP) services with "srvctl commands". See also

the Oracle® "Administration and Deployment Guide".

The DTP service must be started on the node on which the instance of the RAC DB system that is primarily assigned to it is running, i.e. the DTP service `racutmS21`, which is primarily assigned to the instance `racutm2`, must be started on the node on which this instance is running.

1. Enter the service in the file `tnsnames.ora` with a net_service_name:

   Example

```
RACUTMS1 =
  (DESCRIPTION =
     (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP) (HOST=server1) (PORT=1521))
        (ADDRESS = (PROTOCOL = TCP) (HOST=server2) (PORT=1521))
     )
     (CONNECT_DATA =
        (SERVICE_NAME = racutmS12.domain_name )
     )
     (FAIL_OVER = ON)
  )
```

2. In the Open string in the start parameters, assign this net_service_name (in this case RACUTMS1) to the operand "SqlNet".

7.3.2.4 Debug parameters

You have the option of logging the XA interface in openUTM for test purposes. The RMXA start parameter DEBUG= is available for this purpose. This parameter must be passed **before** the other RMXA start parameters. It is not passed to the database system.

> ¡ It is recommended that you only use this function for test purposes, as the output can become very extensive. Every transaction is logged in the task, irrespective of whether there is any access to the database.

The DEBUG= parameter has the following format:

```
.RMXA DEBUG={ YES | ALL },OUTPUT={ SYSOUT | FILE }
```

Explanation

DEBUG= Activates the debug function.

Use DEBUG= as the first RMXA start parameter.

YES Logs the individual XA calls and, for each call,

- the service number
- the transaction counter
- the return value

ALL In addition to the values logged with DEBUG=YES, the status values and the XID are also logged.

OUTPUT= Specifies the output destination.

SYSOUT The log is written to SYSOUT.

FILE Output is sent to a file. The file name has one of the following formats:

`KDC.TRC.XA.`*`appliname.tsn`*
Name of the application, up to 8 characters in length.

tsn
TSN of the UTM task (4-digit)

Alternatively, a file can be assigned using the link name KDCXA.

Example:

```
/SET-FILE-LINK LINK-NAME=KDCXA,FILE-NAME=XA-DEBUG.BSP.01
```

You can enable or disable logging of the XA interface during execution of the application using the administration functions. To do this, use the program interface, the administration tools WinAdmin/WebAdmin or the administration command KDCDIAG XA-DEBUG=. For details, refer to the openUTM manual “Administering Applications”.

7.3.3 Termination of a UTM database application

Normal termination of a UTM database application

A UTM database application is terminated using UTM administration functions; see section“Terminating a UTM application normally via administration”. The database handler is not shut down in this case.

Abnormal termination of a UTM database application

A UTM database application can be terminated abnormally as a result of errors or by the administrator; see section “Terminating a UTM application abnormally”. Following an abnormal application termination, the database and KDCFILE may be in an inconsistent state.

In this case, the data consistency is checked and, if necessary, restored by the subsequent warm start of the UTM database application. In this case, openUTM completes a shared recovery phase with the affected database systems.

7.4 Operating a UTM database application

The operation of a UTM database application is based on the same principles as the operation of a UTM application without a database. The special points to observe are described in the following sections.

7.4.1 User sign-on and sign-off

A user who wants to work with a UTM database application signs on using the client-specific sign-on process for openUTM. The same applies to sign-off.

If a sign-on service is used for signing on, the following must be noted:

- If the user signs on via a terminal or terminal emulation, database calls are not permitted in the first part of the SIGNON service for security reasons, unless this is explicitly permitted at UTM generation with the KDCDEF statement SIGNON, ...RESTRICTED=NO.
- In the second part of the sign-on service, the authorization profile for the user is read from the database, provided the database supports this option. This means that a universal DB/DC authorization concept can be implemented.

More details on sign-on and sign-off can be found in the chapter “Working with a UTM application”.

7.4.2 SAT logging

The following only applies if the database is connected over the IUTMDB interface, i.e. is does not apply to XA databases such as Oracle.

SAT logging can be defined individually for each transaction code (= job-driven) and each UTM user (= user-driven). UTM then logs all the security-related events within UTM and announces activation of SAT logging on the interface to the DB system.

The XA interface is excepted from SAT logging.

Security-related events within the DB transaction must be logged by the DB system if this is permitted. UTM only records whether or not the entire transaction was executed successfully.

More details on SAT logging can be found in the section “User-driven SAT logging” and the section “Job-driven SAT logging”.

7.4.3 Accounting

The following only applies if the database is connected over the IUTMDB interface, i.e. is does not apply to XA databases such as Oracle.

DB systems that do not run in UTM tasks can use the IUTMDB interface to notify openUTM of the resources utilized for processing DB calls. The usage values supplied by the DB system (CPU, I/Os) are stored in the accounting record in the accounting phase; see the section “Structure of an accounting record”.

These values are also supplied to openSM2. As the DB system may not be able to determine this data precisely, evaluation of the data is only possible to a limited extent for measurement purposes, although trends can be identified. The KDCMON event monitor can be used to record exact values from the DB system; see below.

In the accounting phase, the values are multiplied with the defined weights and added to the accounting unit counter; see the section “Structure of an accounting record”.

The usage values are supplied to openUTM by the DB systems SESAM/SQL and UDS/SQL. SESAM/SQL only supply the usage values to openUTM if the gathering of accounting information is enabled in the DBH; see also the manual “SESAM/SQL Database Operation”.

7.4.4 Performance control

Database calls from the program units can only be recorded for the IUTMDB interface and not for to XA databases such as Oracle.

To control the performance of the UTM application, the UTM event monitor KDCMON can be implemented with the formatting tool KDCEVAL. Some of the lists formatted with KDCEVAL also contain database-specific information:

- The TASKS list contains the proportion of time spent on DB calls in the `database` column.
- The KCOP list specifies how often DB measurement data was written in NOOP.
- The TACLIST list specifies the number of DB calls for each TAC.
- The TRACE and TRACE2 lists are trace lists containing both the UTM calls and DB calls.

More details on KDCMON and the associated evaluation lists can be found in the section“KDCMON - UTM event monitor”

7.4.5 Diagnostics

UTM messages, error codes and dumps are important sources of information for diagnosing errors. Database-specific information is also provided for UTM database applications. This information should be examined first if an error could relate to a fault in the database connection. The following UTM diagnostic information is supplied:

- the database-specific UTM messages K068 and K071
- the start error codes of message K049
- messages from the XA database connection K201 through K233
- the incompatible return code KCRCDC
- the DB Diagarea of the UTM dump, if a UTM dump was created
- the CDUMP error codes KDCDBxx, if a CDUMP was created

More details can be found in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

8 Working with a UTM application

This chapter describes the various dialog types with which a terminal user can sign on to and sign off from a UTM application. Communication always following the same principle for all clients:

1. Sign on to the UTM application
   A user can only sign on via clients for which LTERM partners, LTERM pools, or OSI-LPAP partners have been generated in the UTM application; see the openUTM manual “Generating Applications”.
2. Call services of the UTM application:
   openUTM offers a separate authorization concept for data access control, see "Authorization concept of openUTM".
3. Enter UTM user commands if necessary.
4. Sign off from the UTM application.

The details of these steps vary depending on the type of client. The following sections describe the options available for the various clients.

UTM user IDs are used for access, provided the application is generated with user IDs. For information on signing on to a UTM application without user IDs, see chapter "Sign-on process without user IDs".

8.1 Sign-on process with user IDs

If an application is generated with user IDs, openUTM runs through a standard sign-on process for the user in accordance with the type of client. For some types of clients, it is also possible to use your own sign-on process instead of the standard one, see section “Sign-on process with sign-on services”.

A user can sign-on using the following client access points:

- terminals ("Standard sign-on process for terminals")
- UPIC clients and TS applications ("Sign-on process for UPIC clients and TS applications")
- via the Internet using HTTP clients
- OSI TP partner ("Sign-on process for OSI TP partner")
- HTTP clients ("Sign-on process for HTTP clients")
- via the internet using WebServices (WS4UTM) ("Sign-on process in the World Wide Web via WebServices (WS4UTM)")
- via the internet using WebTransactions ("Sign-on process in the World Wide Web via WebTransactions")

Under certain circumstances, it is also possible for several users to sign on under the same user ID, see section “Multiple sign-ons under one user ID”.

8.1.1 Standard sign-on process for terminals

The user must carry out the following steps in order to work with a UTM application via a terminal:

1. Establish a connection to the UTM application

   The establishment of the connection can be initiated either by the UTM application or by the user.
2. Sign on to a UTM application, see the section “Standard sign-on dialog”.

Only then can the user start services and carry out interactive processing, see section“Calling UTM services”.

Connection setup by the user via terminal emulation

A user can also work with a UTM application from a Windows PC by calling a emulation (for example, MT9750) and opening the session configured for this application.

When you set up a session for MT9750, specify the name of the UTM application and the name and/or IP address of the computer on which the application runs. Details can be found in the MT9750 product manual.

8.1.1.1 Standard sign-on dialog

The standard sign-on dialog is always performed when the following two conditions are met:

1. Automatic KDCSIGN (= automatic sign-on check) is not generated for the terminal (see "Automatic KDCSIGN").
2. No sign-on service is generated for the standard application name (generated in MAX APPLINAME) under which the user signed on (see "Sign-on process with sign-on services").

In the standard sign-on dialog, openUTM carries out a sign-on check (system access control). In the sign-on dialog, UTM messages in line mode ask the user for identification. In the UTM generation, you can choose between various variants and also modify the UTM message texts. However, this is the extent to which this procedure can be modified.

Different levels of freedom can be defined for the sign-on check. An overview of all options can be found in the diagram in the section “Scenarios for the UTM sign-on check”.

openUTM outputs the following message to initiate the sign-on check:

```
K002 Connected to application <appliname> - please sign on
```

The user must verify his or her authorization by entering the following details:

`KDCSIGN` *userID* `[,`*password*`]`

The password can be entered in the KDCSIGN command if no password blanking is generated for this user ID (KDCDEF statement USER...,PASS=).

openUTM checks the user ID and the password (if there is one). If the result is negative, then the user is requested to re-enter the data for KDCSIGN. If the user is generated with a password, but the password has not been specified in the KDCSIGN command, openUTM requests the password in a blanked-out field in a second dialog step.

Blanking the password

If password blanking is generated for the user ID (KDCDEF statement (USER...,PASS=(*password*,DARK)), openUTM outputs a UTM message to ask the user to enter the password in a blanked-out field.

In this case, the user has the option of entering a new password to replace the previous one, provided the minimum validity period defined in generation allows the password to be changed at this time. In this case, the user must confirm the new password by repeating it. openUTM checks the old password entered and, if necessary, the new password. If the old password is incorrect, then the user is requested to re-enter the data for KDCSIGN. If the new password was not entered identically both times, a UTM message is output to inform the user and request that the data be reentered.

Validity period of the password

When generating the user ID, you can define a maximum and minimum validity period for the password:

USER ...,PROTECT-PW=(...,*maxtime*,*mintime*).

The minimum validity period means that the user cannot change his or her password until this period has expired.

The maximum validity period means that the user must change the password within the specified period.

If the password will become invalid within the 14 days following the sign-on procedure, openUTM warns the user in a K-message as long as a change is allowed at this time according to the minimum validity period for the password. A password can be changed as described under “Blanking the password”.
To prevent users that have not worked with the application for a long time from forgetting to change their password and then consulting the system administrator, the UTM application can be configured such that these users may sign on one more time after their password has expired, see section “Grace sign-on”.

When the password is changed, openUTM checks the following:

- whether the new password differs from the old one if a maximum validity period has been generated for the password.
  If a password history is generated (SIGNON ...,PW-HISTORY=n), the new password is also checked against the last n passwords.
- whether the new password corresponds to the level of complexity generated for the user ID (USER ...,PROTECT-PW=).
- whether the length of the password is greater than or equal to the generated minimum length (USER ..., PROTECT-PW=).

If the password fulfils all of these conditions, openUTM changes the password. The validity period of the new password again corresponds to the generated value.

If the new password does not satisfy one of these conditions, the following UTM message is output to ask the user to reenter the KDCSIGN information using the old password:

```
K097 Input for new password cannot be used - please sign on
```

If the validity period of the password has already expired when the sign-on attempt is made and if no grace sign-on is generated, the sign-on attempt is rejected with the following UTM message:

```
K120 Password expired - please sign on
```

It is not possible to sign on at the UTM application under this user ID again until the UTM administrator has assigned a new password to the user ID.

Grace sign-on

If the validity of the password has already expired when the user attempts to sign on and if the application is generated with grace sign-on (SIGNON ...,GRACE=YES), a K message informs the user that his or her password is no longer valid. The user is also asked to enter the old password and a new password. To change the password, a password must be reentered in the corresponding blanked-out field.

KDCSIGN with ID card

If the insertion of an ID card (magnetic strip card) is prescribed for the user ID in the UTM generation, openUTM outputs a UTM message asking the user to insert an ID card into the ID card reader. When this message is output, the keyboard of the terminal is locked. The lock is not released until the ID card is inserted. If no ID card is available, the keyboard can alternatively be unlocked with the K14 or ESC: key.

openUTM checks the ID card information, i.e. openUTM compares the character string defined in the KDCDEF statement USER ...,CARD=(...) with the corresponding character string on the ID card. If the result is negative, UTM message K031 is output to inform the user and request that the KDCSIGN information be reentered. If the result is positive, the user can work with the application.

The ID card must remain in the ID card reader for as long as the user wishes to work under this user ID. If the ID card is removed prematurely, openUTM detects this with the next dialog input or before, and implicitly generates KDCOFF. See also section “Signing off from a UTM application”.

> If the ID card is removed, the same message is output as when you press function key K14. If the user is working under a user ID which requires KDCSIGN with ID card check, the assignment of K14 (KDCDEF statement SFUNC) to a TAC or return code makes no sense for this user ID. If the KDCOFF command is assigned to function key K14, then, for all user IDs, entering K14 has the same effect as KDCOFF .

Scenarios for the UTM sign-on check

The following diagram shows the possible variants of the UTM-sign-on check, depending on the KDCDEF generation. If incorrect data is entered, openUTM outputs a specific UTM message and asks the user to reenter the information. If several unsuccessful sign-on attempts are made in succession from a particular terminal or under a particular user ID, openUTM outputs UTM message K094 with the standard destination SYSLOG (system log file). The maximum permitted number of unsuccessful sign-on attempts before UTM message K094 is initiated can be defined in the UTM generation with the KDCDEF statement SIGNON ... SILENT-ALARM=. An MSGTAC program unit can respond to this UTM message.

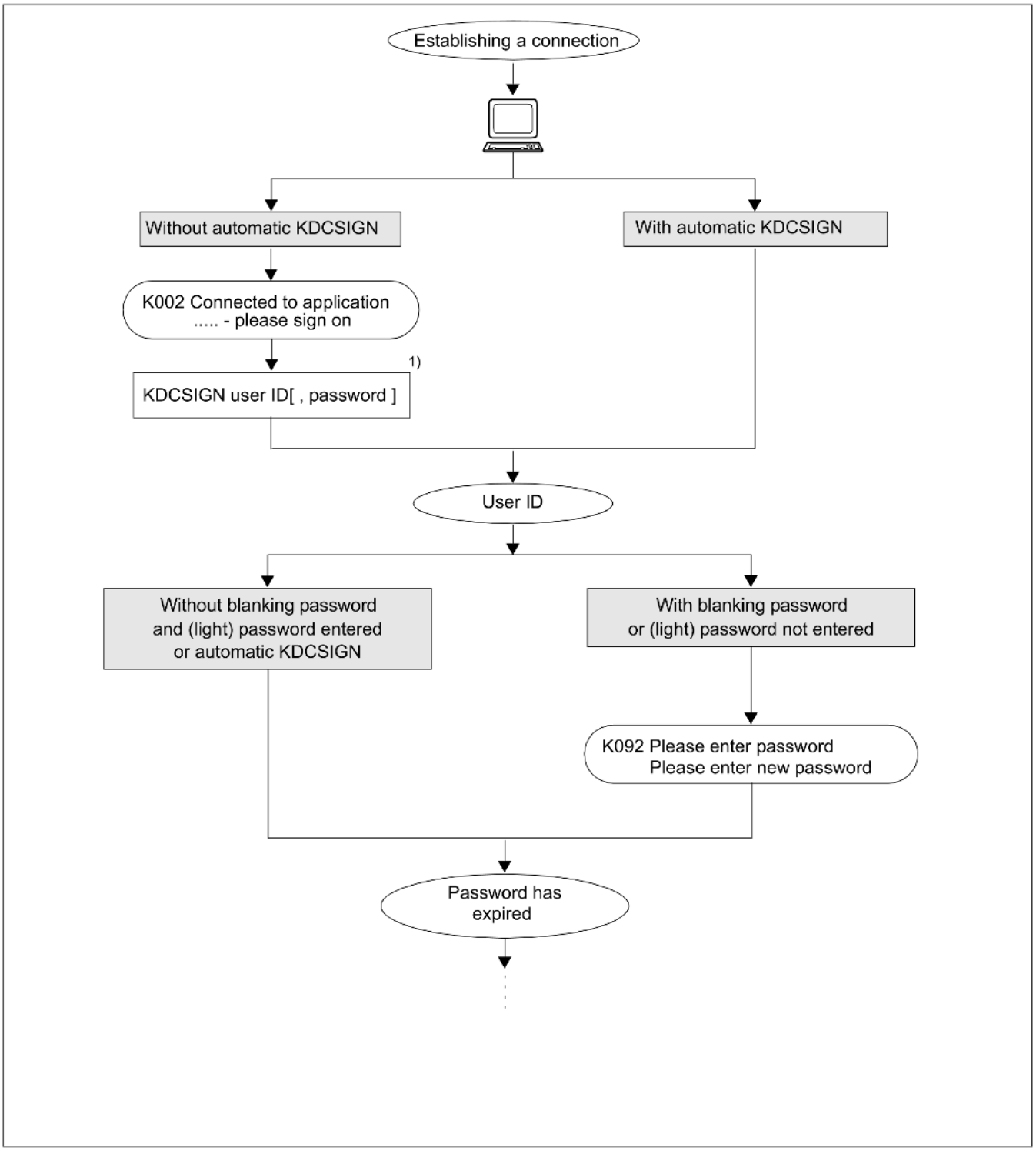


Figure 5: Sign-on check scenarios for applications with user IDs (part 1)

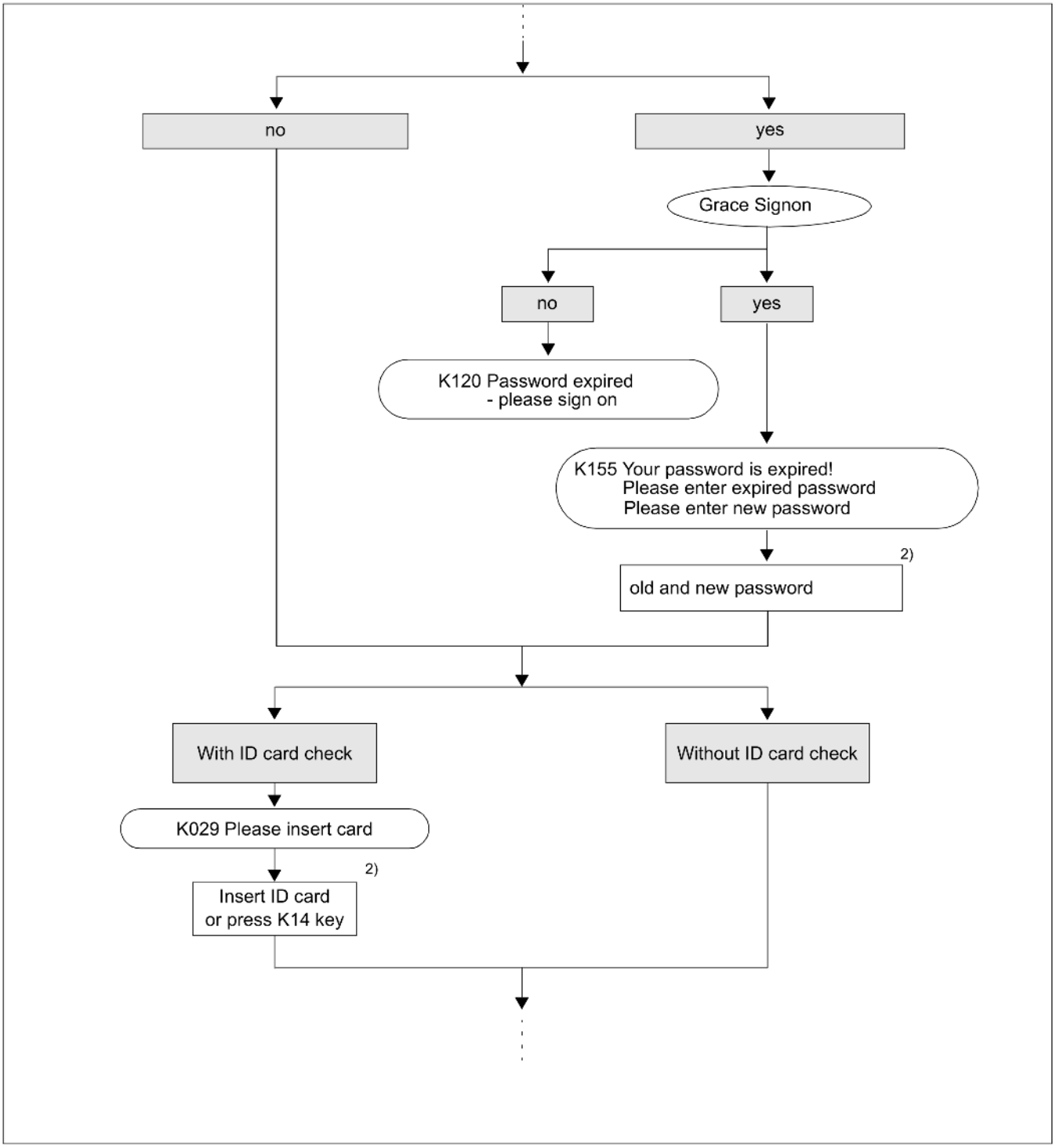


Figure 6: Sign-on check scenarios for applications with user IDs (part 2)

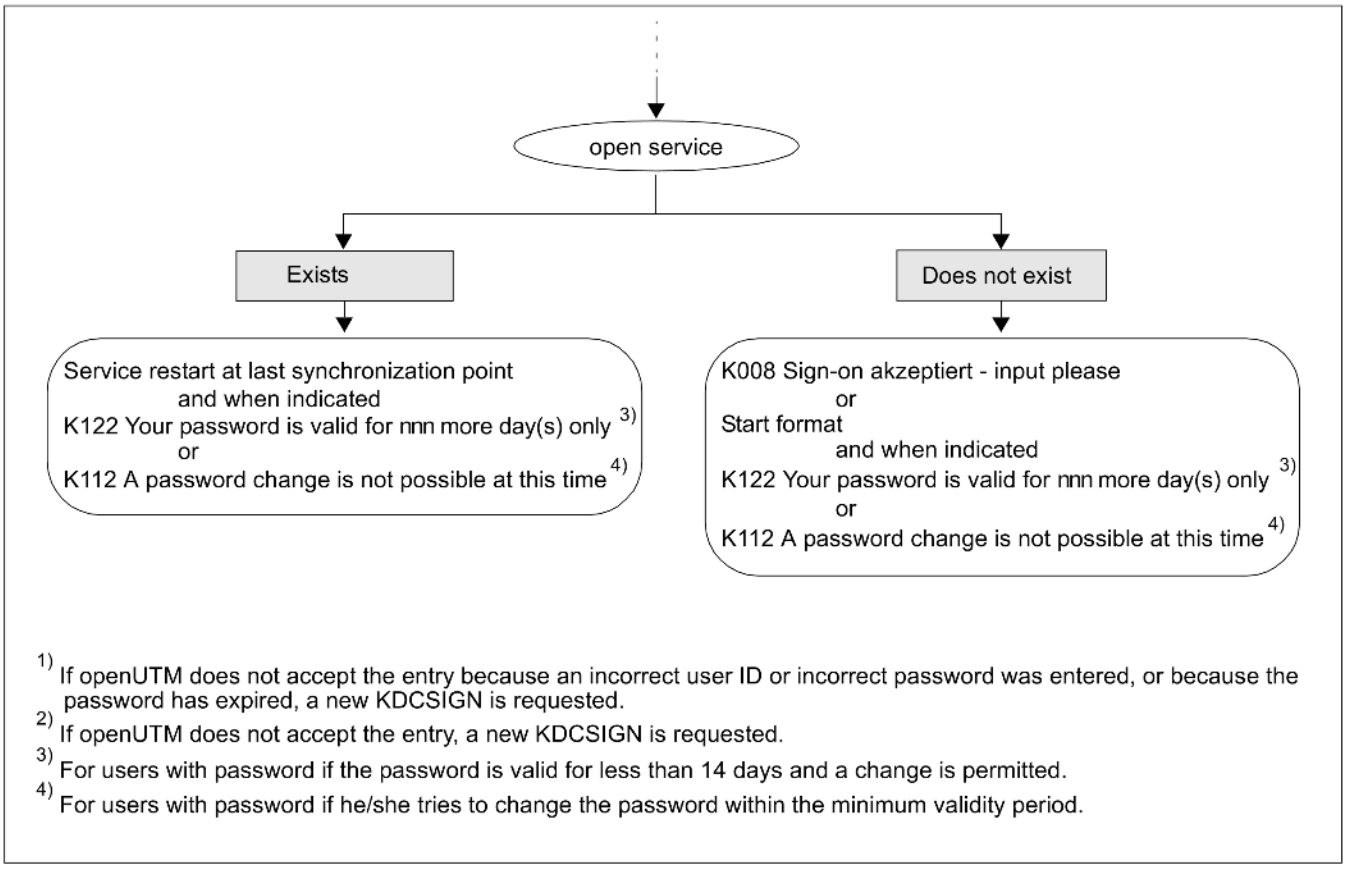


Figure 7: Sign-on check scenarios for applications with user IDs (part 3)

8.1.1.2 Automatic KDCSIGN

If the KDCDEF statement LTERM...,USER=*username* was specified for a terminal, after the connection setup openUTM behaves as though the user had already verified his or her authorization, i.e. KDCSIGN need **not** be entered. If an ID card or blanked-out password must be entered for this user ID, openUTM requests this input from the user.

After KDCOFF BUT has been entered, it is also possible to work at this terminal under a different ID (see section “Signing off from a UTM application”).

8.1.2 Sign-on process for UPIC clients and TS applications

UPIC clients and TS applications are clients that were generated with partner type UPIC-R, APPLI or SOCKET, and in case of partner type SOCKET communicate via UTM Socket Protocol (USP).

The connection is set up by the client in the case of UPIC clients, and by the client or openUTM in the case of TS applications; connection setup by openUTM is only possible if the TS application is generated explicitly with a PTERM statement.

If the client sets up the connection, the client must know the name or, in the case of partner type SOCKET, the port number of the UTM application as well as the host name and/or host address.

When a connection is set up successfully, a UPIC client or TS application signs on in two stages:

1. Implicit sign-on using a **connection user ID**

   A connection user ID is strictly assigned to an LTERM partner of a TS application or a UPIC client and is created explicitly or implicitly at UTM generation:

   - Explicit creation by USER= specification in the LTERM statement.
     Additional characteristics can be defined with the KDCDEF statement USER for connection user IDs defined in this way.
   - Implicit creation by openUTM if no USER was specified in the LTERM statement or if an LTERM pool is used (TPOOL statement). The LTERM name is then used as the name of the connection user ID; in the case of an LTERM pool, the LTERM name is made up of the generated prefix and a serial number, e.g. UPIC0025. For LTERM pools, special key codes can be assigned to the connection user ID with TPOOL ... USER-KSET=. The access options of the connection user ID can thus be restricted.

   If no sign-on attempt is made under a real user ID, the preliminary sign-on of the connection user ID becomes permanent. This is recorded with a message. In the case of UPIC clients, this message is also output if the client subsequently signs on under a real user ID.

Explicit sign-on using a

real user ID (optional)

UPIC clients and TS applications behave differently in this case:

- In the case of UPIC clients, the user ID and sign-on data must be set in the respective UPIC interface calls. UPIC then transfers these values to openUTM, which then performs the sign-on for the transferred user ID. This replaces the connection user ID for the duration of the conversation.

  At the end of the conversation, the user is signed off again if the UTM application is generated with SIGNON OMIT-UPIC-SIGNOFF=NO.
  If the UTM application is generated with SIGNON OMIT-UPIC-SIGNOFF=YES, the user remains signed on after the end of the conversation and is only signed off,

  - if another user ID is passed in the UPIC protocol before a new UPIC conversation is started over the same UPIC connection,
  - or when the connection is cleared.

If the UPIC client does not transfer any sign-on data in the UPIC interface calls, signing on using a real user ID is only possible with a corresponding sign-on service; see "Sign-on process with sign-on services".

- A user can only sign-on under a real user ID via a transport system connection if an appropriate sign-on service is generated for the application; see "Sign-on process with sign-on services". It is not possible to sign on with a real user ID using the standard sign-on dialog.

  If a TS application signed on using a real user ID, this user ID replaces the connection user ID for the full duration of the connection.

In the case of both UPIC clients and TS applications, the connection user ID remains signed on for at least as long as the real user ID. If the connection is lost, a renewed connection setup attempt may be rejected if a program is still running under the real user ID and the connection user ID is thus also considered to be signed on. In this case, the user must wait until the program terminates before signing on again.

8.1.3 Sign-on process for OSI TP partner

In order for an OSI TP partner to sign on to the UTM application, the partner must know the address of the OSI TP access point of the UTM application. This data is configured in the OSI TP partner.

In the case of OSI TP partner, the connection setup initiative can come from either the partner or openUTM. Thereby several parallel connections, known as associations, can be established via one logical connection. An association name is assigned to each association.

Following a successful connection setup, the client first signs on under its association name. This name is made up of the name specified in OSI-LPAP ...,ASSOCIATION-NAMES= and a serial number, e.g. ASSOC03.

Once an application context containing the abstract syntax UTMSEC has been generated for OSI TP communication between the two partners (in case of openUTM in the APPLICATION-CONTEXT parameter of the OSI-LPAP statement), the communication partner can pass a real user ID and authorization data in the relevant protocol fields. openUTM then signs on with the user ID that has been passed. This sign-on then applies for the duration of the OSI TP dialog. The user is then signed off again at the end of the OSI TP dialog.

If no real user ID is passed, the client remains signed on under its association name.

8.1.4 Sign-on process for HTTP clients

HTTP clients are clients that were generated with PTYPE=SOCKET and communicate using the HTTP protocol.

For an HTTP client to be able to log on to the UTM application, it must know the address of the transport system endpoint of the UTM application. With HTTP clients, the initiative to establish a connection always comes from the client, and an HTTP client sends an HTTP request to the transport system endpoint.

Once the connection has been successfully established, an HTTP client is logged on in two steps:

1. Implicit logon using a **connection user ID**
   A connection user ID is permanently assigned to the LTERM partner of an HTTP client and is generated explicitly or implicitly during UTM generation:
   - Explicitly by specifying USER= in the LTERM statement.
     Additional properties can be defined for a connection user ID defined in this way using the KDCDEF statement USER.
   - Implicitly by openUTM, if no USER was specified in the LTERM statement or if it is a LTERM pool (TPOOL statement). The LTERM name is then used as the name of the connection user ID; for a LTERM pool, the LTERM name is made up of the generated prefix and a sequential number, for example, HTTP0025. For LTERM pools, special key codes can be assigned to the connection user ID with TPOOL ...USER-KSET=. This allows you to restrict the access options of the connection user ID.

   If no login under a real user ID follows, the temporary login of the connection user ID becomes a final one. This is logged with a message.
2. Explicit logon using **a real user ID** (optional)
   In an HTTP request, an HTTP client can send authentication data to the UTM application in the Authorization header field. If this header is specified, the value must begin with "basic " and <userid>:<password> must be base64 encoded. Otherwise the request will be rejected with status code "400 invalid http header format". openUTM checks the passed authentication data. If the check fails, openUTM rejects the login with "401 user not authorized". Otherwise the user is logged on and can call the services of the application, see "Starting services from the HTTP client".

   The UTM application may also require the HTTP client to send authentication data. This can be configured either for all HTTP requests received via a transport system endpoint by setting the USER-AUTH parameter in the KDCDEF statement BCAMAPPL to *BASIC, or for the path specification of an HTTP request by setting the USER-AUTH parameter in the KDCDEF statement.

8.1.5 Sign-on process in the World Wide Web via WebServices (WS4UTM)

A service of the UTM application can be called from WebService clients using WS4UTM. This allows the user to access certain services of a UTM application over the Web.

The WebService client can be used to structure the sign-on process via WS4UTM:

1. The user specifies a WebService name and a method in his/her WebService client. The WebService is permanently linked to a UTM application by the configuration. A connection to the UTM application is established over UPIC.
   The WebService client may possibly execute an intermediate dialog, for instance to obtain proof of authorization.
2. As when using a terminal, the user may have to specify their UTM user ID and password. Whether or not the user has to go through an authorization dialog of this type and the appearance of any such dialog will depend on the way in which the WebService client is structured. It is, for instance possible to "hide" the UTM user ID /password in the WebService client or to specify it within the configuration of the WebService, with the result that the authorization dialog is handled internally.
3. The job data (TAC and user data) is sent together with the authorization data to a WebService server via http /Soap and then to the UTM application over the UPIC connection. The UPIC connection is cleared again after the response has been returned to the WebService client.

The Apache Axis server is used as the WebService server.

Communication takes place over Apache Tomcat and Axis using Soap messages and the http protocol. WS4UTM uses the UPIC interface in openUTM in order to connect to the UTM application.

For further information, refer to the manual "Web-Services for openUTM".

8.1.6 Sign-on process in the World Wide Web via WebTransactions

A UTM application can be connected to the World Wide Web via WebTransactions. This means that a user can access the services of a UTM application using a browser.

Signing on via WebTransactions is simple and can be configured using the WebTransactions application:

1. The user enters the URL of the WebTransactions application in the browser.The connection is then established to the UTM application. The WebTransactions application might output an intermediate dialog box, e.g. to verify authorization for accessing the WebTransactions application.
2. The user may have to specify the UTM user ID and password (if necessary), as with a terminal. However, the actual configuration of the WebTransactions application determines the format of the sign-on dialog and whether or not the user must complete such a dialog. For example, it is possible to “hide” the UTM user ID /password in the WebTransactions application so that the sign-on dialog runs internally and the user is signed on immediately the URL is entered.

The user can then call the services of the application, see "Calling UTM services".

To connect to the UTM application, WebTransactions uses either the terminal interface or the UPIC interface of UTM.

 More details can be found in the WebTransactions manuals “Web Access to openUTM Applications via UPIC” and “Web Access to OSD Applications”, if the terminal interface is used.

8.1.7 Multiple sign-ons under one user ID

If the user ID was generated at KDCDEF generation with RESTART=NO and the UTM application with default value MULTI-SIGNON=YES, a user can be signed on to the UTM application via different connections – though only once via a connection to a terminal. Multiple sign-ons are only possible for real user IDs, **not** for connection user IDs. More details on connection user IDs can be found on "Sign-on process for UPIC clients and TS applications".

If a user signs on under a user ID generated with RESTART=YES via an HTTP client or an OSI TP partner with the functional unit “commit” selected for its conversation, a further sign-on is possible under this user ID because openUTM does not restart the service in this case and the user ID thus behaves as though no restart was generated.
The same applies if the user signs on via an OSI TP partner and executes an asynchronous request.

Otherwise, a user can only be signed on once at any one time under a user ID generated with RESTART=YES, because the resources needed to restart the service are assigned to the user ID.

Preventing multiple sign-ons for user IDs with RESTART=NO

The MULTI-SIGNON parameter of the SIGNON statement can be used at UTM generation to define that a user can only be signed on to openUTM once at any one time regardless of the restart attribute.
However, this definition does not apply to sign-ons via an OSI TP partner for the execution of asynchronous requests.

8.1.8 Sign-on process with sign-on services

Sign-on services, also known as SIGNON event services, are user-programmed services that can be used to define your own sign-on processes. Sign-on services can be used by terminals, UPIC clients, and TS applications, i.e. by clients generated with the PTERM or TPOOL statement (except HTTP clients).

Calling sign-on services

A sign-on service is linked to the application name. If a client signs on under a particular application name, the sign-on service associated with this application name is started and replaces the standard sign-on process described in the preceding sections. If several application names are generated with BCAMAPPL statements, several different sign-on services can exist in an application. This means that client-specific sign-on services can be created, e.g. one for terminals, one for UPIC clients, and one for TS applications. More details can be found in sections “Sign-on service for terminals” through “Sign-on service for UPIC clients”.

If no sign-on service is generated for an application name, the client runs through the standard sign-on process.

Generating sign-on services

Sign-on services are generated as follows; see also the openUTM manual “Generating Applications”:

- TAC KDCSGNTC is used to generate the sign-on service for the standard application name (defined in MAX APPLINAME).
- BCAMAPPL *appliname2...,*SIGNON=*signon-tac* is used to generate the sign-on service for the application name *appliname2*. *signon-tac* must be defined in a TAC statement.
- If you also want UPIC clients to be able to use sign-on services, SIGNON ...,UPIC=YES must be generated as well.

A PROGRAM statement is also needed for each of these TACs. The name of the first program unit run through in the sign-on service is specified here.

Programming sign-on services

The special KDCS calls SIGN ST, SIGN ON and PEND PS are used for programming a sign-on service. A detailed description of how to program a sign-on service and what rules to observe in the process can be found in the corresponding section of the openUTM manual „Programming Applications with KDCS”.

8.1.8.1 Sign-on service for terminals

A sign-on service for terminals is generally made up of two parts:

Sign-on service	
Part 1	Part 2
The service asks the user for identification, reads the authorization data with MGET, and transfers this data to openUTM for checking. The service is **not** yet assigned to any user ID.	openUTM has accepted the authorization data and has assigned the sign-on service to the established user ID.

Between the first and second part of the sign-on service, openUTM can - if necessary - insert an intermediate dialog that is controlled by openUTM messages. This intermediate dialog reads a magnetic strip card or a password generated with USER...,PASS=(...,DARK). In these cases, openUTM asks the user to enter this information, as in the standard sign-on dialog. This procedure offers the users additional security for accessing their applications. In the intermediate dialog, it is possible (as in the standard sign-on dialog) to replace the previous password by a new one.

Special cases of the sign-on service for terminals

The sign-on service must be changed accordingly for the UTM generation of LTERM partners with automatic KDCSIGN and for signing on via OMNIS.

LTERM partners with automatic KDCSIGN

The sign-on service receives the information that the user ID is already known to the system when calling SIGN ST. An intermediate dialog can now be run to query a magnetic strip card or a password.

Signing on via OMNIS or multiplex connection (OMNIS)

When the user signs on to the UTM application via multiplex connection, then OMNIS transmits the authorization data on to openUTM via the PUTMMUX protocol for further verification. If the data is correct, then openUTM starts the sign-on service that contains the appropriate information from the SIGN ST call. If the data is incorrect, then the sign-on service is not started and the distributor must inform the user on the terminal.

8.1.8.2 Sign-on service for TS applications

The sign-on service is started under the connection user ID.

The authorization data of a real user ID can be passed in the sign-on service via the SIGN ON call. If openUTM accepts the data, then the user is signed on under the specified user ID when the sign-on service ends properly. The sign-on attempt is rejected if the authorization data of the TS application is incorrect or if there is an open service under the connection user ID.

If the sign-on is unsuccessful under a real user ID, a successful sign-on under a real user ID must follow within the same sign-on service, as otherwise the connection is cleared down when the sign-on service is terminated. This means that the connection user ID is not a fallback step for a failed sign-on attempt.

If there is no user ID passed in the sign-on service, then the user is signed on permanently under the connection user ID when the sign-on service terminates properly.

8.1.8.3 Sign-on service for UPIC clients

A distinction is drawn between two possible scenarios when signing on using a sign-on service:

- The UPIC client transfers authorization data to openUTM in the UPIC protocol. If openUTM accepts the data, the sign-on service is started under the transferred real user ID and the client is signed on under this user ID, provided the sign-on service is completed successfully.
- If the UPIC client does not transfer any authorization data in the UPIC protocol, the sign-on service is started under the connection user ID. The authorization data of a real user ID can be passed in the sign-on service. If openUTM accepts the data, then the user is signed on under the specified user ID when the sign-on service ends properly. If no authorization data is passed, then the conversation runs under the connection user ID.

If the sign-on fails under a real user ID, a successful sign-on must follow under a real user ID, as otherwise the conversation is terminated when the sign-on service is terminated. This means that the connection user ID is not a fallback step for a failed sign-on attempt.

To ensure that client programs can be implemented regardless of whether or not the UTM application uses a sign-on service, messages from the client that were not read in the sign-on service can be read in the subsequent program unit after terminating a program unit of the sign-on service with PEND PA, PEND PR, PEND PS or PEND FC without preceding MPUT.

8.1.8.4 Possible applications for the sign-on service

The sign-on service offers the user a range of practical options, which are outlined below:

- The sign-on service for terminals can be conducted as a formatted dialog.
- Start formats can contain current data such as the date, time, a bulletin, etc.
- In an application with multilingual users, LTERM-specific start formats can be used to output a welcome screen in the respective language. This option is only available for terminals.
- User-specific start formats are useful for showing information specific to a user group (bulletin, menu, etc.).
- TS applications can sign on to a UTM application using a sign-on service with a real user ID. They are thus integrated in the system access and data access concept of openUTM.
- A real name entered by the user can be converted into a user ID which is defined in the UTM generation (USER *username*).
- In the case of a global DB/DC authorization concept, a database call can be used in the second part of the sign-on service to retrieve the current authorization profile for this USER from the database and possibly store it in a user-specific long-term storage area (ULS).
- In the second part, the sign-on service can ask the user to change his or her password, for example because the system is monitoring the time span in which the user may use the same password.
- Statistics can be produced on all attempted and successful sign-ons.
- The FHS format handling system offers the option for terminals of loading the P keys using a global attribute of a #format. In the second part of the sign-on service, for example, the P keys can be loaded user-specifically.
- The sign-on service can also provide the user with useful information in the case of a subsequent service restart. Such information includes bulletins, maps of keyboard layout, or a display of the service restart. This requires an additional dialog step.
- If openUTM starts the sign-on service following a SIGN OB call (= KDCOFF BUT by program), it may be advisable to read the last input from the terminal with MGET if new authorization data was already entered there.

8.1.8.5 Properties of sign-on services

Outputting the last dialog message by the sign-on service

If there is not a service restart pending and the sign-on service is terminated with MPUT PM and PEND FI, the last dialog message of the user‘s last session is output if the user is generated with RESTART=YES. The user can then continue working with the same screen that was being used when the last session was terminated, regardless of whether this occurred inside or outside of a service.

Messages

If a UTM application uses a sign-on service, then the following messages are not produced (and therefore not output to the SYSLOG and MSGTAC):

K001, K002, K004 through K008.

Message K033 (Successful sign-on) is also output when a sign-on service is used.

Unsuccessful attempts in the sign-on service

In the sign-on service, unsuccessful attempts of the user to sign on can be intercepted: if openUTM does **not** accept the authorization data entered by the user, the sign-on service can ask the user to repeat the input. The maximum number of input attempts can be programmed. If this number is exceeded, the sign-on service should terminate. UTM then shuts down the connection in the case of TS applications and terminals, whereas only the conversation is ended in the case of UPIC.

In addition, openUTM counts all of the unsuccessful attempts of a client or unsuccessful attempts under a user ID made in uninterrupted succession, also over a series of sign-on services. The maximum permitted number of failed sign-on attempts must be defined in the UTM generation. After this number of failed sign-on attempts has been made (see openUTM manual “Generating Applications”, KDCDEF statement SIGNON, operand SILENT-ALARM), openUTM reports this event to SYSLOG (silent alarm, UTM message K094). Sign-on attempts by unauthorized persons can be uncovered and averted with an MSGTAC routine.

Abnormalities in the sign-on service

openUTM checks whether the rules for the sign-on service are observed. This also provides protection against any manipulation of the program units of the sign-on service. If such errors occur, openUTM terminates the sign-on service with PEND ER and shuts down the connection to the terminal. The connection is then shut down in the case of TS applications and terminals, whereas only the conversation is ended in the case of UPIC.

8.1.8.6 Sample programs for the sign-on service

In conjunction with openUTM, program units supplied as COBOL source programs implement a complete sign-on service with formatted interface to the terminal. This sign-on service is suitable for all generation variants. The format used contains English texts.

This template can be modified to suit the user’s needs, thereby providing an easy way of producing a sign-on procedure with a formatted interface to the user. In this way, there is no need for the user to begin the programming from scratch.

8.1.9 Behavior in the event of locked clients/LTERM partners

Behavior for locked clients

Clients can be locked by UTM generation (PTERM...,STATUS=OFF) or administration command. Locking a client has the following effects:

- Any connection setup request will be rejected.
- Any existing connection will be retained; the lock only comes into effect if a new connection setup request is received from this client.

Behavior for locked LTERM partners

LTERM partners can be locked by UTM generation (LTERM...,STATUS=OFF) or administration command.

In the case of UPIC clients and TS applications, locking the LTERM partner has the same effect as locking the client.

In the case of terminals, locking an LTERM partner has the following effects:

- Any connection setup request will be carried out, but the following UTM message will be output after the connection has been established:

  ```
  K027 Terminal <ltermname> is locked - contact administrator  or sign off.
  ```

- Any existing connection will be retained; the next input from the terminal will be acknowledged with UTM message K027.

8.2 Sign-on process without user IDs

openUTM does not perform a sign-on check for UTM applications for which no user IDs are generated. The clients are signed on under their LTERM names or association names. UPIC clients, HTTP clients and OpenCPIC clients are not permitted to transfer user IDs in this case.

If the UTM application uses sign-on services ("Sign-on process with sign-on services"f), an application-specific sign-on check can then be performed, e.g. using a database with authorization data.

If sign-on services are not used, the user can work with this application as soon as a connection has been successfully established to the UTM application. In the case of terminals and TS applications (clients with partner type APPLI or SOCKET that communicate via the UTM Socket Protocol (USP) in case of partner type SOCKET), the user receives a message from openUTM depending on whether an open service is still known for this LTERM partner.

- If no open service is known for the LTERM partner in the application, openUTM outputs the UTM message

  `K001 Connected to application <example> - input please`

  In the case of terminals, the start format for this LTERM partner is output, if generated. The user can then start services and enter UTM user commands.
- If an open service is known for this LTERM partner in the application, the output from the last synchronization point of the interrupted service is displayed on the screen and the user can continue the service. See also “Service restart” and “Screen restart” in the openUTM manual „Programming Applications with KDCS”. One of the prerequisites here is that RESTART=YES was generated for this LTERM partner. However, this also means that the user may also be able to continue the service of another user.

Note that openUTM links a service to the LTERM partner in an application without user IDs. An interrupted service can therefore only be continued from the same client, unless the assignment of LTERM partner and physical client (defined in the PTERM statement) is changed accordingly with the administration command KDCSWTCH.

If clients are locked, the behavior is the same as for user IDs; see "Behavior in the event of locked clients/LTERM partners".

> **CAUTION!**
>
> In a UTM application without user IDs, all users have administration authorization.

8.3 Calling UTM services

If the UTM sign-on check runs successfully, the user is authorized to work with the UTM application, i.e. he or she can start new services (see below) or continue open services.

Sections “Starting services from the terminal” to “Starting services from TS applications” illustrate how new services are started for the individual client types. For a description of what happens when an open service is still known for this user ID in the application, see section “Service restarts”.

8.3.1 Starting services from the terminal

Following a successful sign-on, the user can start a service by entering a transaction code (TAC) or pressing an appropriately generated function key.

Starting a service by entering a transaction code

If no sign-on service is performed, users may find themselves in the following situations:

- openUTM outputs the following message in line mode:

  ```
  K008 Sign-on accepted - input please
  ```

  The user can start a service by entering a TAC and possibly a message. The first eight characters input are interpreted by openUTM as the TAC. If the TAC is shorter than 8 characters, it must be separated from the message by a blank.
- If a start format was generated for the user, the appropriate fields of the start format must be completed. The field with the TAC, also called the control field, does not necessarily appear first.

If a sign-on service is performed, the sign-on service determines the next step. The user then receives output in format or line mode, or a service is started immediately.

Starting a service using a function key

A TAC can be assigned to a function key at UTM generation, e.g. F10; see KDCDEF statement SFUNC. When the user presses this key, the service is started regardless of whether the user is working in line mode or is in a start format.

Entering invalid transaction codes

If the user enters an incorrect TAC, the following message is output:

```
K009 Transaction code <tac> is invalid - input please
```

If a BADTACS dialog service is generated in the application, then the BADTACS service is started instead. After the BADTACS dialog service has ended, the user remains signed on and can start a service as described above.

8.3.2 Starting services from the UPIC client and OSI TP partner

After the connection has been set up, the OSI TP partner or UPIC clients can start conversations. To this end, the TAC is set by the client, e.g. using the *Set_TP_Name* function on the CPI-C interface or a corresponding entry in the side information file. This TAC is transferred to openUTM, possibly in conjunction with authorization data. When the sign-on check has been performed successfully, the following apply:

- In the case of OSI TP partner and UPIC clients with no sign-on service, the service associated with the TAC is started immediately.
- In the case of UPIC clients with a sign-on service, the service associated with the TAC is not started until the sign-on service has been concluded.

The user is signed off again at the end of the conversation if he or she signed on for this conversation under a real user ID. This is not valid for UPIC clients in applications generated with SIGNON OMIT-UPIC-SIGNOFF=YES, see "Sign-on process for UPIC clients and TS applications".

8.3.3 Starting services from the HTTP client

After the connection has been set up, the HTTP clients can start conversations. For details specifying the authorization data and the TAC please refer to the openUTM manual "Concepts and Functions" and to the openUTM manual "Programming Applications with KDCS".

8.3.4 Starting services from TS applications

TS applications (clients with partner type APPLI or SOCKET that communicate via the UTM Socket Protocol (USP) in case of partner type SOCKET) behave similarly to terminals:

- If there is no sign-on service, the TS application receives message K001 if the message destination PARTNER was assigned to this message; see the description of the KDCMMOD tool in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

  The TS application can then start a service by transferring a TAC, and possibly a message, to the UTM application. In this case, the first 8 characters of the message are interpreted as the TAC. If the TAC is shorter than 8 characters, it must be separated from the message by blanks.
- If a sign-on service is performed, this service determines the next step. The sign-on service can either start a service directly to send a message to the TS application. In the latter case, the next message must contain a TAC in the first 8 characters, i.e. the same applies as when no sign-on service is used (see above).

Once the service has terminated, the next service can be started.

8.3.5 Service restarts

A service restart is generally performed if:

- a client signs on under a user ID that was generated with RESTART=YES,
- and an open service is still known for this user ID in the application.

If a message was sent to the client at the last synchronization point then openUTM sends this message to the client again. The user can then restart the service. Otherwise the open service is continued immediately.

Depending on the type of client and on the sign-on process involved, the following apply to the service restart:

- Standard sign-on process for terminals and TS applications:
  openUTM performs the service restart automatically.
- Standard sign-on process for UPIC clients and OSI TP partner:
  The client must start a specific conversation, which requests the restart using the UTM user command KDCDISP (see the manual „openUTM-Client for the UPIC Carrier System”, for example). A service cannot be restarted from OSI TP clients if the “commit” functional unit was selected.If an open service exists for the user but no service restart is requested the open service will be terminated abnormally.
- Signing on using a sign-on service:
  The sign-on service must initiate the restart or terminate the open service abnormally.

> ¡ In an application with user IDs, a service is linked to the user ID. This means that the user can continue an interrupted service even on a different client, provided the LTERM partner of the client has the correct authorization and the client type remains the same.

8.4 Authorization concept of openUTM

In addition to system access control based on user IDs, openUTM offers a sophisticated system access and data access concept. This makes it possible to control which users can access which services of the UTM application via which LTERM partners.

You can choose between a user-oriented variant (**lock/key code** concept) and a role-oriented variant (**access list** concept). These variants are generated using lock codes, access lists, keysets, and key codes:

- A service is protected either with lock codes (lock/key code concept) or with an access list (access list concept) (TAC statement LOCK= or ACCESS-LIST=).
- A user ID receives a keyset with one or more key codes (USER statement KSET=). The key codes define the authorizations.
- An LTERM partner receives a keyset with one or more key codes, as well as lock codes if the lock/key code concept is used (LTERM or TPOOL statement, KSET= and LOCK= operands).
- Keysets are defined separately in KSET statements.

The preconditions under which users can sign on and when they can start or continue a service (following a service restart) are outlined in the following table for both concept variants.

Action	**Preconditions**	
	Lock/key code concept	**Access list concept**
Sign on via specific LTERM partner	A key code of the user ID matches the lockcode of the LTERM partner.	Sign-on is always possible.
Start a service	The user ID and LTERM partner have a key code that matches the lockcode of the TAC.	The user ID and LTERM partner each have a key code which is contained in the access list of the TAC. The key codes of the user ID and LTERM need not be identical.
Continue service (following service restart)	A key code of the LTERM partner via which the user continues the service must match the lockcode of the follow-up TAC.	A key code of the LTERM partner via which the user continues the service must be contained in the access list of the follow-up TAC.

Messages in the event of incorrect authorization

If authorization is invalid, the following messages may be output to the terminal user (a corresponding return code is supplied with the sign-on service):

`K005 User identification <`*user*`> is locked - please sign on`

If the key code of the user does not match the key code of the LTERM partner (sign-on service: return code U02).

`K009 Transaction code <tac> is invalid - input please`

If the user or LTERM is not authorized to start the service. If a BADTAC service is generated, the BADTAC service is started instead.

```
K123 LTERM does not have the rights to continue the service - please sign on
```

If the LTERM partner via which the user signed on at the service restart is not authorized to start the follow-up TAC (sign-on service: return code U16). This message may be output in particular if a user continues the service from a different terminal and hence a different LTERM.

> More information can be found in the openUTM manual “Concepts and Functions” and the openUTM manual “Generating Applications”.

8.5 Signing off from a UTM application

The following sections describe the various ways in which a client can sign off from the UTM application or is signed off by UTM. In this case, terminals differ from all other clients because users can only sign off from the application explicitly from terminals.

Signing off in the event of a timeout

Maximum wait times can be defined at UTM generation using:

- the TERMWAIT= (PEND KP timer) and PGWTTIME= (PGWT timer) operands in the KDCDEF control statement MAX
- the IDLETIME= (transaction end timer) operand of the PTERM or TPOOL statement or the OSI-LPAP statement for OSI TP partners.

If a wait time set with these timers expires, the following message is output to terminals:

```
K021 No input within the specified period
```

openUTM then signs off the user ID and shuts down the connection to the client. The client can subsequently sign on to the application again and continue a possibly open service, see section “Service restarts”.

Signing off with the KDCOFF command

A terminal user can sign off from the UTM application by entering the UTM command KDCOFF or KDCOFF BUT. See also the UTM user command KDCOFF on "KDCOFF - sign off from a UTM application".

KDCOFF from a program

openUTM offers the function calls SIGN OF and SIGN OB, which can be used to trigger the effect of the KDCOFF or KDCOFF BUT user command in a dialog program unit. SIGN OF/OB is possible for terminals, UPIC clients, and TS applications. These calls are not permitted in program units running for an OSI TP partner.

SIGN OF and SIGN OB work as follows:

SIGN call	Command	Effect
SIGN OF	KDCOFF	openUTM shuts down the connection to the client
SIGN OB	KDCOFF BUT	The connection remains open for terminals; the user is signed off. In the case of UPIC clients and TS applications, the connection is shut down (as with SIGN OF).

The call has different effects for terminals and UPIC clients/ TS applications:

- In the case of terminals, openUTM first outputs the MPUT message and message K095 to the terminals. Only with the next (arbitrary) input from the terminal is the user signed off and the

  connection shut down (with SIGN OF).

- In the case of UPIC clients and TS applications, the MPUT message is sent and the connection shutdown is then initiated immediately.

Some of the possible applications of the SIGN OF/OB function call are outlined below:

- Applications with particular security requirements. After signing off, a user can only process a single service.
- Improved data protection: The screen can be overwritten by the last MPUT; no service-specific data remains on the screen.
- The control part of the screen also offers “Sign Off” or “Sign On” as possible follow-up actions. Depending on the input, the follow-up program unit then creates a SIGN OF or SIGN OB call. Following the dialog output of this program unit and the subsequent input, either the connection to the terminal is shut down or the sign-on service is started.

8.6 UTM user commands for terminals

This section describes all of the UTM user commands available to the terminal user after signing on (the KDCSIGN command needed to sign on is described on "Standard sign-on dialog"):

- KDCFOR, for outputting the basic format
- KDCOUT, for requesting asynchronous messages
- KDCDISP, for requesting the last dialog message again
- KDCLAST, for repeating the last output
- KDCOFF, for signing off

The KDCDEF statement SFUNC can be used when generating the application to assign UTM user commands to K/F keys; these commands can then be entered by pressing the corresponding function keys.

8.6.1 KDCFOR - output the basic format

The KDCFOR command can be used to output a basic format. A basic format is used for entering the data for a job on the screen in format fields.

By entering KDCFOR { *format-identifier*[1] | '*format-identifier*'[2] }, the format is output at the terminal with the specified format identifier. This format applies as the basic format of the user until another KDCFOR command with a different format identifier is entered.

By entering the KDCFOR command without a *format-identifier*, the basic format which is currently valid can be output at the terminal.

> ¡ It is **not** possible to output #formats with the KDCFOR command.

In the event of errors, openUTM outputs one of the following UTM messages:

`K013 Error in command KDCFOR - input please`

The KDCFOR command does not have the prescribed form.

`K014 No base format defined - input please`

The user has specified a KDCFOR command without a format identifier, but no basic format is known as yet.

`K015 Formatting error ..... - input please`

An error occurred in the attempt to output the basic format.

`K003 Command KDCFOR is not permitted at this time.`

The KDCFOR command is not permitted on this level of the dialog (with the UTM application).

---

[1] if *format-identifier* = *

[2] if *format-identifier* = *,+,-

8.6.2 KDCOUT - output asynchronous messages

With the KDCOUT command, the user can request the output of asynchronous messages. At generation time, the ANNOAMSG={ YES | NO } operand in the LTERM or TPOOL statement defines how openUTM is to output asynchronous messages to this terminal (LTERM) or LTERM pool (TPOOL).

Output with ANNOAMSG=Y

In this case, openUTM announces asynchronous messages with the following UTM message:

```
K012 nnn asynchronous message(s) present
```

The UTM message appears in the system line together with a dialog output at this terminal. The number of asynchronous messages is specified with *nnn*. The user can retrieve these messages using the KDCOUT command. If there are no messages for the terminal when KDCOUT is entered, openUTM outputs the UTM message:

```
K020 No message(s) present
```

When an asynchronous message is retrieved with KDCOUT, it is deleted by the next input, except when KDCLAST is entered (see "KDCLAST - repeat the last output").

If the user is in a format-driven dialog with a service and outputs an asynchronous message with KDCOUT, the last output format is thereby destroyed. To continue the service, the user must redisplay the output format with the user command KDCDISP (see "KDCDISP - output the last dialog message"). If this is not done, openUTM automatically executes a KDCDISP. The user must then repeat the input.

The result of the KDCDEF statement LTERM ..., RESTART= NO is that any pending asynchronous messages are deleted when the connection is set up or shut down to this LTERM partner.

The function variants of openUTM have the following effects on the handling of asynchronous messages:

- With UTM-S applications, asynchronous messages are logged even if the application run is interrupted and are retained until retrieved with KDCOUT.
- With UTM-F applications, asynchronous messages are only stored during the application run. They are lost when the application run terminates.

Output with ANNOAMSG=N

If a client is connected to an LTERM partner generated with LTERM...,ANNOAMSG=N or TPOOL ..., ANNOAMSG=N, the KDCOUT command is not permitted on this client. ANNOAMSG=N means that openUTM outputs asynchronous messages immediately at the terminal, i.e. without prior announcement. openUTM therefore rejects the KDCOUT command at this terminal with UTM message K003:

```
K003 Command KDCOUT is not permitted at this time.
```

In a UTM application with user IDs, asynchronous messages are output at the earliest after the sign-on check has been concluded successfully.

If you want to continue your dialog with the application after the asynchronous message has been output, the screen of the last dialog output must be requested beforehand with KDCDISP. Otherwise, openUTM executes an automatic KDCDISP. This automatic screen restart is only suppressed by openUTM if the asynchronous message was output with the same format as the last dialog output.

> An automatic restart is **always** implemented with #formats.

8.6.3 KDCDISP - output the last dialog message

While a UTM application is running, a user can output the last dialog message once again with the KDCDISP command.

If the user enters the KDCDISP command after the sign-on service has concluded or after returning from an inserted service, openUTM redisplays the last screen of the last session or the last screen of the interrupted service.

For information on screen output in FHS-DE formats, see also the section “KDCDISP, KDCLAST in FHS-DE formats” (KDCLAST - repeat the last output).

The KDCDISP command is useful in the following situations:

- As a result of transmission problems or operating errors at the terminal, the screen content after a dialog output is partially or fully destroyed.
- The user has received asynchronous messages on the screen while processing a service (either requested with KDCOUT or sent automatically by openUTM) and then wants to continue the open service. In this case, the KDCDISP command is issued to redisplay the last dialog output.

  If the user wants to continue the service directly in the asynchronous format message, openUTM implicitly creates a KDCDISP and the input must be repeated.
- When the UTM application has been terminated and restarted, the user can (for orientation purposes) issue the KDCDISP command to repeat the last dialog output of the service concluded before the application terminated. However, this only applies with a UTM-S application and if the service restart facility was not explicitly deactivated by the KDCDEF statement USER ...,RESTART=NO (or LTERM ...,RESTART=NO, if the application was generated without user IDs).

8.6.4 KDCLAST - repeat the last output

The KDCLAST command enables you to repeat the last output message at the terminal, regardless of whether this was a dialog message or an asynchronous message. For information on screen output in FHS-DE formats, see also the following section.

If the last message output was an asynchronous message, this output is repeated with KDCLAST. However, the asynchronous message is thereby not yet released.

If the KDCLAST command is entered after the sign-on service has concluded, openUTM redisplays the last screen of the sign-on service. If the command is entered after returning from an inserted service, the last screen of the inserted service is redisplayed.

KDCDISP, KDCLAST in FHS-DE formats

It is advisable to use the FHS-DE function RESHOW to redisplay the last screen output in FHS-DE formats.

In the FHS-DE intermediate dialog, the user must issue the FHS-DE function RESHOW to request the repetition of the last screen output. The RESHOW function can only be called using a K key assigned with RESHOW in the KEY format belonging to the format. This key must not be generated with SFUNC in openUTM. See also the User Guide “FHS - Format Handling System for openUTM, TIAM, DCAM”.

If the UTM commands KDCDISP and KDCLAST are specified in FHS-DE formats, they are only processed by openUTM if they are specified in the first input field or in the UTM control field of an FHS-DE screen format. No field in the screen format can be filled such that an FHS-DE intermediate dialog would be run as a follow-up action. If these conditions are fulfilled, the command is transferred from FHS-DE to openUTM for processing and as a result the last screen output is repeated.

If, on the other hand, one of the commands is specified in a dialog box (in the FHS-DE intermediate dialog) or in a screen format followed by an intermediate dialog, then KDCDISP and KDCLAST do not cause openUTM to repeat the last screen output. FHS-DE first processes the intermediate dialogs. In this case, FHS-DE does not recognize KDCLAST or KDCDISP.

8.6.5 KDCOFF - sign off from a UTM application

You can enter the UTM command KDCOFF to sign off from the UTM application. The connection to the terminal is thereby shut down. You can then establish another connection to a UTM application from this terminal. If you sign off at the end of a transaction while a service is being processed, the processing is interrupted. It can be continued when you later sign on to the UTM application again if the user is generated with RESTART=YES.

KDCOFF BUT

By entering KDCOFF BUT, you can sign off in such a way that the connection between the terminal and the UTM application is retained. It is needed for a subsequent sign-on, or the sign-on service is started.

Messages

If KDCOFF [BUT] is entered, openUTM responds by outputting one of the following UTM messages:

`K019 Sign-off for application <appliname> accepted`

The user entered KDCOFF or, in an application without user IDs, entered KDCOFF BUT. The terminal is no longer connected to the UTM application.

`K018 Sign-off for application <appliname> accepted - please sign on`

The user entered KDCOFF BUT in an application with user IDs and without a sign-on service. openUTM asks the user to sign on again.

`K003 Command KDCOFF is not permitted at this time.`

The command is entered after a PEND KP call or blocking call (e.g. PGWT) of the program unit.

9 Exchanging programs during operation

openUTM offers functions for exchanging application programs or parts of an application program during operation. openUTM uses the interfaces and functions of the BLS for the exchange operation.

The application must have been generated with at least one LOAD-MODULE statement if program exchange is to be possible.

The following can be exchanged during operation using UTM administration functions:

- All non-shareable application parts that are not linked statically.
- All application parts in a common memory pool valid for a user ID (MPOOL..., SCOPE=GROUP). However, this requires that no more than one openUTM application is connected to this common memory pool.
- The complete application.
- A nonprivileged subsystem in LLM format, for which the private slice is generated as a load module with LOAD-MODE=STARTUP | ONCALL. A system administration command is required in addition to the KDCPROG command. Please note that a consistency gap exists for nonprivileged subsystems in OM format. See also the information in section “Shared code in system memory”.

The following cannot be exchanged during operation:

- Program components that were linked statically to the application program.
- Program components that were loaded as shareable programs in a common memory pool generated with SCOPE=GLOBAL.
- Program components in common memory pools that were loaded with SCOPE=GROUP and to which several UTM applications are connected under the same user ID.
- Load modules in which additional user-specific modules other than the one generated with a LOAD-MODULE statement are loaded by the autolink function along with the load module.
- Load modules containing TCBs.
- Load modules containing runtime modules. These can only be exchanged with the entire application program.
- The load module that contains the KDCADM administration program unit.

The UTM administration functions for exchanging programs are described in the openUTM manual “Administering Applications”.

9.1 Linking and generating

To be able to exchange application parts, the start LLMs must be linked beforehand as a link load module (LLM) and be made available in a program library as type L element (see section “Linking LLMs”).

The rest of the program units must also be linked beforehand as a link load module (LLM) and be made available in a program library as type L elements, or they must be contained as object modules (OMs) in an object module library (OML) or in a program library as type R elements.

Exchangeable load modules must always be generated in a LOAD-MODULE statement with LOAD-MODE `not equal` STATIC. If they are loaded in a common memory pool, this must be generated with MPOOL ,... SCOPE=GROUP. If program units or data areas are incorporated in a load module, the LOAD-MODULE operand must be defined as appropriate in the associated PROGRAM or AREA statements.

9.2 Exchanging application parts

The exchange of application parts must be explicitly requested using the administration command KDCPROG. In this case, you must specify details about the version of the new load module to be loaded. openUTM checks whether the specifications are permissible and initiates the program exchange. In the program exchange process, openUTM does not verify whether the assignment defined with KDCDEF in the LOAD-MODULE, AREA, and PROGRAM statements corresponds to the actual division of the load modules in the libraries.

The changes that resulted from the administrative action to exchange programs in the application program loaded are saved by openUTM beyond the end of the application run, i.e. the versions of the load modules that have been modified by the administration actions will be loaded during the next start. The version numbers of the exchanged load modules can be transferred to the new KDCFILE even in the case of an update generation using KDCUPD, which means that the modules last loaded are reloaded at the next application start.

In the program exchange, only a program component loaded with **one** load procedure can be exchanged as a single module, i.e. **only one** LLM or OM can be exchanged as a part. Only the module which is generated by openUTM is exchanged, not the entire load unit; in other words, if the load unit contains parts of the runtime system that were loaded with autolink, these are not unloaded in the exchange.

Example

The load module A-LLM is contained in the library OWN-LIB and is generated with:

```
LOAD-MODULE A-LLM,LIB=OWN-LIB                                    -
                 ,VERSION=001                                    -
                 ,LOAD-MODE=STARTUP                              -
                 .ALTERNATE-LIBRARIES=YES
```

A-LLM contains a program unit APU, for example, which calls a function *bfunc*. This function is in B-LLM, which is contained neither in A-LLM nor in the library OWN-LIB. In this case, A-LLM is loaded at the start of the application program with B-LLM by the autolink function. When A-LLM is exchanged with KDCPROG, B-LLM remains in the memory with *bfunc*. This can lead to inconsistencies if B-LLM contains subroutines from application programs and not just runtime modules. If B-LLM also contains application logic, A-LLM and B-LLM are to be linked to create one LLM.

When exchanging individual load modules, the event exits SHUT and START are not executed. These are only activated when the entire application program is terminated and loaded dynamically as a result of a program exchange in a task.

The exchange process runs differently depending on when (STARTUP or ONCALL) and where (common memory pool or not) the load module is loaded.

9.2.1 Exchanging a load module with LOAD-MODE=STARTUP

When exchanging a load module that was generated with LOAD-MODE=STARTUP, the program run in the tasks of the application is not terminated. Instead, the relevant load module is unloaded and then a new version of this load module is loaded. The program exchange can be implemented simultaneously by several tasks of an application. During the exchange, different states of the application program are loaded in the tasks of the UTM application. Each task of the application implements the requested program exchange after processing the current job. The program exchange is concluded when the new version of the load module is loaded in each task of the application.

Until the program exchange is concluded in all tasks, no further program exchange can be started. The administrator can use the administration command KDCINF SYSP to ascertain whether or not a program exchange is running at present.

The version specifications of the old and new load module may be the same.

When the program exchange is concluded successfully openUTM generates UTM message K074, which is output to SYSOUT. The UTM message can, however, also be evaluated using an MSGTAC program, in order to make this information accessible to the administrator.

If openUTM has to abort the program exchange, openUTM generates UTM message K075.

9.2.2 Exchanging a load module with LOAD-MODE=ONCALL

If you want to exchange a part of an application which was generated in a load module with LOAD-MODE=ONCALL, only the new version number to be loaded for the respective load module will be entered in the UTM tables when the administration command KDCPROG is processed.

The load module of the new version will not be loaded by each task of the application until the next time a program unit of this load module is called in the task. This program exchange can be implemented simultaneously by several tasks of an application. Until the requested program exchange has been implemented by all tasks of the UTM application, different states of the application program are loaded in the individual tasks. However, it is ensured that each task implements the requested exchange before another program unit is activated which is contained in the load module to be exchanged.

The exchange of an ONCALL load module does not have a blocking effect on subsequent commands for program exchange; in other words, immediately after processing the KDCPROG command the administrator can initiate a new program exchange with another KDCPROG command. However, the program library must not be modified after the administration call has been issued, as otherwise the program exchange may result in errors.

If the version numbers of the new and the old load module are the same, no program exchange is implemented.

9.2.3 Exchanging a load module in a common memory pool

Load modules that are fully or partially contained in common memory pools can first of all be marked for exchange with the KDCPROG command.

These load modules are not exchanged until the entire application is subsequently exchanged (see below).

If the application terminates before the application exchange, then the load modules *remain* marked for exchange. The new (remain marked) versions are loaded during the next restart.

If a program exchange is requested using KDCPROG, details about the version of the new load module to be loaded must be specified. The new version and the old version must be different.

Program units that contain both shareable and non-shareable parts should be linked as LLM, because otherwise there will be two load modules in OM format that cannot be exchanged simultaneously (consistency problem!).

9.3 Exchanging the entire application

You can use the command KDCAPPL PROG=NEW to exchange the entire application.

At least one LOAD-MODULE must have been generated for the application if it is to be possible to exchange the entire application.

Exchanging the entire application may be useful or necessary in the following situations:

- Program components in common memory pools are to be exchanged.
- Program components generated with LOAD-MODE=ONCALL are to be unloaded.
- Programs in common memory pools were added dynamically, and these programs are to be loaded for the application.

When the entire application is exchanged, each task of the application is unloaded in succession and then loaded dynamically. In the dynamic loading process, the new versions of the load modules are loaded. For load modules generated with *HIGHEST-EXISTING for the version, the highest existing version is loaded.To minimize the interruption to the operation of the application, openUTM exchanges only one task of the application at any one time.

The administrator can issue the command KDCINF SYSP to ascertain whether or not a program exchange is running at present.

When the program exchange is concluded successfully, openUTM generates UTM message K074, which is output to SYSOUT. The UTM message can also be evaluated using an MSGTAC program in order to make this information accessible to the administrator.

If individual load modules cannot be loaded or if all programs specified in the UTM generation are not linked in the load module, this does not abort the application exchange. If an unavailable program is called at a later stage, openUTM generates a PEND ER.

If openUTM has to terminate the program exchange, openUTM generates UTM message K075. As an insert, UTM message K075 contains the TSN of the task that aborted the application exchange. In the SYSOUT file of this task, BLS messages and openUTM messages may indicate the cause of the abort. One possible reason for aborting an application exchange may be that an AREA in a newly loaded module is not available. If the application exchange is aborted, a task of the UTM application may be terminated. However, the other tasks of the application continue to run. The load modules marked for exchange remain marked and, as soon as the problem has been rectified, the application exchange can be initiated again.

9.4 Adding programs dynamically

Amongst other things, dynamic administration allows for the regeneration of programs while the application is running. For more details on dynamic administration, see the openUTM manual “Administering Applications”.

Before these programs can be called, they must first be loaded. In this case, the program must be linked to the assigned load module and must be made available with a new version in the program library specified in the LOAD-MODULE statement when generating.

The administrator must then exchange this load module with the KDCPROG command or by a program call. If the load module is contained in a common memory pool, the entire application must be exchanged, i.e. first of all the module is marked for exchange (e.g. with KDCPROG) and then the entire application is exchanged (e.g. with KDCAPPL PROG=NEW).

10 Fault tolerance of openUTM

Fault tolerance in this context means that a UTM application can still remain operational when errors occur in individual program units that force openUTM to abort a transaction. openUTM then ensures that the application program is terminated and reloaded so that the error does not spread any further and have a negative effect on other users of the application and their data.

With regard to the error behavior of openUTM, a distinction is made between:

- Internal UTM errors and errors in the system environment

  These errors result in an abnormal termination of the application, just like the administration command KDCSHUT KILL or when issuing a KDCADMI call with operation code KC_SHUTDOWN and subcode KC_KILL. openUTM creates a UTM dump for each process of the application. The UTM dump is edited using the UTM tool KDCDUMP. A description of this procedure can be found in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.
- Errors in the application program

  These are errors in program units. They can be divided into two groups:
  - errors that lead to the reloading of the application
  - errors that permit the program to continue.

10.1 Errors detected by openUTM

A program unit is terminated abnormally by openUTM in the following situations:

- A PEND ER or FR was programmed.
- A UTM call supplied a KDCS return code >=70Z. In this case, openUTM internally sets PEND ER.

In both situations, openUTM aborts the service. If a PEND FR was programmed, then openUTM does not take any other action.

If the service was terminated by a PEND ER (in a program or internally), then openUTM creates

- a UTM dump with REASON=PENDER, which only conveys the data of KDCROOT.
- a memory dump of the class 5 and class 6 memory. This can be analyzed using the BS2000 utility DAMP.

  By default, this memory dump is suppressed because openUTM runs as an ENTER process. If you want the memory dumps of the class 5 and class 6 memory to be created, specify the following command in the start procedure (see "Starting the application") before the application program is loaded:

  ```
  /MODIFY-TEST-OPTIONS DUMP=YES
  ```

openUTM then terminates the affected application program. This prevents follow-up errors arising due to the possible overwriting of an application program.

With a branch statement in the start procedure, the application program can be loaded dynamically and openUTM can continue running with the desired number of tasks.

> **CAUTION!**
>
> If there is no branch statement, the task is terminated and, in the case of the last task, the application is terminated.

10.2 Errors detected by the BS2000 system which lead to a STXIT event

In openUTM, STXIT routines are defined for the following event classes:

- PROCHK
- TIMER
- ERROR
- ABEND
- TERM

If the STXIT events PROCHK, TIMER, and ERROR occur, openUTM provides the following alternatives to the language connection module of the program unit last active:

- output diagnostic information and terminate the program unit abnormally, or
- continue the program unit as defined (e.g. with ON-Condition in PL/I)

If STXIT event ABEND or TERM occurs, openUTM sets PEND ER, terminates the application program with TERM and writes the following to SYSOUT:

- the register contents at the time of the event
- the instruction counter at the time of the interrupt
- the interrupt weight

If the program units last active were COBOL, Assembler or SPL program units, all STXIT events are also handled like TERM and ABEND.

For more information on STXIT events, see the BS2000 manual “Executive Macros”. A table indicating the assignment of event to interrupt weight can be found in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

User-defined STXIT routines

Separate STXIT exits can be assigned to the PROCHK and ERROR events in the program units. In this case, separate administration blocks must be created for these events with the STXDNEW operand.

These user-defined STXIT routines **must** be terminated with the EXIT CONTINU=YES macro call so that openUTM can start its own (and last) STXIT routine of the same event class.

11 SAT logging

Security-related UTM events can be logged using the BS2000 function SAT (**S**ecurity **A**udit **T**rail). SAT is used to audit unauthorized infiltration attempts, for example, thereby facilitating an immediate response to such events (alarm function). Any possible damage can thus be minimized or avoided altogether.

The prerequisites for implementing SAT logging are the BS2000 component SECOS and the subsystem SATCP. openUTM allows you to control SAT logging of UTM events for your application using the KDCDEF generation (see openUTM manual “Generating Applications”, MAX statement) and using UTM SAT administration functions (see "UTM SAT administration commands").

For a UTM application (generated with MAX ...,SECLEV=NO), SAT logging can be switched on in the UTM generation (MAX ...,SAT=ON) or using UTM SAT administration functions (KDCMSAT SAT=ON). With UTM SAT administration functions, logging can be switched off again at any time during operation (KDCMSAT SAT=OFF). When SAT logging is switched on, minimum logging is implemented.
Minimum logging covers the following UTM events:

- A task signs on to or off from the UTM application
- A UTM SAT administration command is entered
- Program components are exchanged using BLS

Other events can also be defined. The logging of these events can be switched on and off for specific events, specific users, and specific jobs. The predefinition of the events to be logged is called preselection (see "Preselection - defining the events to be logged"). Preselection can take place in the UTM generation and using UTM SAT administration functions.

The structure of SAT log records is described in the Appendix on "Structure of SAT log records".

> **CAUTION!**
>
> The BS2000 safety representative (on shipment, this is the BS2000 user ID $SYSPRIV) can suppress SAT logging (see the BS2000 manual “SECOS”). The SAT administrator of the UTM application must therefore coordinate with the BS2000 safety representative. In the absence of coordination, the message K126 is output; see the manual openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”.

11.1 Security-related UTM events

Eleven defined security-related UTM events can be logged. A result is logged for each event: success or failure. Success, for example, means that when a user signs on, openUTM accepts the specified user ID and authentication data.

The table below indicates which UTM events exist and the result that causes the event to be logged.

Name of event	Meaning	Logging possible
TASK-ON	Connect a task to the UTM application	Success
TASK-OFF	Disconnect a task from the UTM application	Success
SIGN	Sign on a UTM user	Success / Failure
CHANGE-PW	Change the user password	Success / Failure
START-PU	Create a job or start a program unit run	Success / Failure
END-PU	Terminate a program unit run	Success
DATA-ACCESS	Access a global service UTM storage area (ULS, GSSB, TLS)	Success / Failure
ADM-CMD	Execute an administration call	Success / Failure
SEL-CMD	Execute a UTM SAT administration command	Success / Failure
CHG-PROG	Exchange load modules using BLS	Success / Failure

Notes

- The event SEL-CMD is the only event also logged if SAT logging is switched off for the UTM application, providing the generation of the BS2000 system permits SAT logging.
- The events TASK-ON, TASK-OFF and CHG-PROG are always logged when logging is switched on for the UTM application (minimum logging).
- The event “end of transaction” (END-PU with TACIDEN=T or C) is always logged when logging for END-PU was not explicitly switched off by event-specific preselection (OFF in the SATSEL statement) and if at least one event was logged for this transaction.
- The other events in the table are only logged if they were defined by preselection and logging is switched on.
- All UTM events can be linked with the ALARM function of SAT. When the event occurs, a UTM message is then output at the console of the BS2000 computer.

11.2 Preselection - defining the events to be logged

The events SIGN, CHANGE-PW, START-PU, END-PU with TACAID=P, DATA-ACCESS and ADM-CMD are not logged automatically, even when logging is switched on. Logging must also be switched on for the specific events. Controlling whether or not these events are logged is called preselection. Logging can be controlled for specific events, specific users, and specific jobs. Preselection can take place in the UTM generation and using UTM SAT administration functions.

If the preselection values are set via the administration, then the following is true:

- The preselection values for event-driven logging are only valid for the duration of the application run. The generated values are used again each time the application is started.
- The preselection values for user and job-specific logging remain in effect, even for UTM-F. You can transfer them to a new generation with an inverse KDCDEF.

11.2.1 Event-driven SAT logging

SAT logging can be switched on and off individually for each of the events SIGN, CHANGE-PW, START-PU, END-PU, DATA-ACCESS and ADM-CMD. For the DATA-ACCESS event (access to UTM storage area), logging can be controlled individually for each of the storage types global secondary storage area, terminal-specific long-term storage area (TLS), and user-specific long-term storage area (ULS).

You can specify the following for each individual event:

OFF	The event is never logged, even when user-specific or job-specific logging is switched on.
SUCC	The event is logged if the result is success.
FAIL	The event is logged if the result is failure.
BOTH	The event is logged regardless of the result.
NONE	No event-driven logging.

The preselection values for event-driven logging can be set in the UTM generation using the SATSEL statement. Using UTM SAT administration functions, you can set the preselection values with the command:

```
KDCMSAT SATSEL=...,EVENT=(...)
```

This setting is only valid for the duration of the application run.

Example

The “change password” event (CHANGE-PW) is to be logged if the result is success (the change was accepted by openUTM) or if the result is failure. In the latter case, the issuing of an administration command is to be logged.

UTM generation:

```
SATSEL BOTH,EVENT=CHANGE-PW

SATSEL FAIL,EVENT=ADM-CMD
```

Administration:

```
KDCMSAT SATSEL=BOTH,EVENT=CHANGE-PW

KDCMSAT SATSEL=FAIL,EVENT=ADM-CMD
```

11.2.2 User-driven SAT logging

For each individual UTM user, you can define whether the SIGN, CHANGE-PW, START-PU, END-PU, DATA-ACCESS, ADM-CMD events initiated by this user and any security-related events of a participating database are to be logged. However, the events are not logged for the user if SAT logging is switched off for this event with OFF:

UTM generation: `SATSEL OFF,EVENT=...`
UTM SAT administration: `KDCMSAT SATSEL=OFF,EVENT=...`

You can specify the following for each user:

SUCC	The events initiated by the user are logged if the result is success.
FAIL	The events initiated by the user are logged if the result is failure.
BOTH	The events initiated by the user are logged regardless of the result.
NONE	No user-driven logging.

The preselection values for user-driven logging can be set in the UTM generation with the statement:

`USER username,...,SATSEL=...`

Using UTM SAT administration functions, the preselection values can be set with the command:

`KDCMSAT SATSEL=...,USER=username`

Any of the values BOTH, SUCC, FAIL or NONE can be specified for SATSEL. These settings are retained past the end of the application run, even for UTM-F.

11.2.3 Job-driven SAT logging

For each individual transaction, you can define whether the CHANGE-PW, START-PU, END-PU, DATA-ACCESS, ADM-CMD events initiated by the associated program unit and any security-related events of a participating database are to be logged. In addition, the creation of jobs of this transaction code (START-PU) is logged. However, an event is not logged for the transaction code if SAT logging is switched off for this event with OFF.

You can specify the following for each transaction code:

SUCC	The events of the program unit run are logged if the result is success.
FAIL	The events of the program unit run are logged if the result is failure.
BOTH	The events of the program unit run are logged regardless of the result.
NONE	No job-driven logging.

The preselection values for job-driven logging can be set in the UTM generation with the following statement:

```
TAC tacname,...,SATSEL=...
```

Using UTM SAT administration functions, you can set the preselection values with the command:

```
KDCMSAT SATSEL=...,TAC=tacname
```

These settings are retained past the end of the application run, even for UTM-F.

11.2.4 Defining the preselection values

You can specify the preselection values in the UTM generation without switching on SAT logging. In this case, the specifications are presettings for SAT logging which can be switched on during operation if required using UTM SAT administration functions. The defined values can also be changed using UTM SAT administration functions. However, changes for event-driven logging only apply for the duration of the current application run.

Presettings in the UTM generation:

```
MAX ...,SAT=OFF
SATSEL ...
TAC ...,SATSEL=...
USER ...,SATSEL=...
```

Activation by UTM SAT administration:

```
KDCMSAT SAT=ON
```

11.2.5 Linking the preselection values

If several preselection values are set for events (event-driven, user-driven, job-driven), the result is the inclusive-OR operation from the individual preselection values.

The tables below indicate the possible combinations of SAT logging conditions.

Meaning of columns:

EVENT Event-specific activation of logging
(UTM generation SATSEL ...;
UTM SAT administration KDCMSAT SATSEL=...,EVENT=...)

TAC Job-specific activation of logging
(UTM generation TAC ...,SATSEL=...;
UTM SAT administration KDCMSAT SATSEL=...,TAC=...)

USER User-specific activation of logging
(UTM generation USER ...,SATSEL=...;
UTM SAT administration KDCMSAT SATSEL=...,USER=...)

Result Result of inclusive-OR operation, i.e. the combination of preselection values.

If the value OFF is generated for an EVENT (event not logged), the value OFF is transferred as the result.

EVENT	SUCC															
USER	SUCC				FAIL				BOTH				NONE			
TAC	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE
Result	SUCC	BOTH	BOTH	SUCC	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	SUCC	BOTH	BOTH	SUCC

EVENT	FAIL															
USER	SUCC				FAIL				BOTH				NONE			
TAC	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE
Result	BOTH	BOTH	BOTH	BOTH	BOTH	FAIL	BOTH	FAIL	BOTH	BOTH	BOTH	BOTH	BOTH	FAIL	BOTH	FAIL

EVENT	BOTH															
USER	SUCC				FAIL				BOTH				NONE			
TAC	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE
Result	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH	BOTH

EVENT	NONE															
USER	SUCC				FAIL				BOTH				NONE			
TAC	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE	SUCC	FAIL	BOTH	NONE
Result	SUCC	BOTH	BOTH	SUCC	BOTH	FAIL	BOTH	FAIL	BOTH	BOTH	BOTH	BOTH	SUCC	FAIL	BOTH	NONE

Example of linking the preselection values

The following preselection values are generated:

Event-specific:

```
SATSEL FAIL,EVENT=(SIGN,TLS)
SATSEL SUCC,EVENT=(CHANGE-PW,GSSB,ADM-CMD)
SATSEL BOTH,EVENT=(START-PU,ULS)
SATSEL NONE,EVENT=END-PU
```

User-specific:

```
USER BSPUSER,SATSEL=FAIL
```

Job-specific:

```
TAC BSPTAC,SATSEL=SUCC
```

The result is therefore the following preselection values. The preselection values are queried using the UTM SAT administration command KDCISAT.

Event-specific with the KDCISAT command:

SAT	USER	TAC	EVENT	RESULT
OFF			SIGN CHANGE-PW START-PU END-PU GSSB TLS ULS ADM-CMD	FAIL SUCC BOTH NONE SUCC FAIL BOTH SUCC

User-specific with the command KDCISAT USER=BSPUSER:

SAT	USER	TAC	EVENT	RESULT
OFF	BSPUSER		SIGN CHANGE-PW START-PU END-PU GSSB TLS ULS ADM-CMD	FAIL BOTH BOTH FAIL BOTH FAIL BOTH BOTH

Job-specific with the command KDCISAT TAC=BSPTAC:

SAT	USER	TAC	EVENT	RESULT
OFF		BSPTAC	SIGN CHANGE-PW START-PU END-PU GSSB TLS ULS ADM-CMD	BOTH SUCC BOTH SUCC SUCC BOTH BOTH SUCC

11.3 Rules for SAT logging

- The events TASK-ON, TASK-OFF, CHG-PROG and, with restrictions END-PU (see section “Security-related UTM events”) are always logged when SAT logging is not switched off for the UTM application.
- All other events are logged when logging is not switched off for the particular event and when at least one of the three criteria of event-specific, user-specific, or job-specific preselection is fulfilled for this event.
- UTM logs an event as successful (SUCC) when it has successfully passed the UTM authorization check and the action can be performed by UTM. Otherwise, the event is logged as failed (FAIL).
- UTM events initiated by KDCS calls (e.g. DATA-ACCESS) are not logged if the function cannot be executed due to an illegal value of KCOM, KCLA, KCLM, etc. (see the openUTM manual „Programming Applications with KDCS“). These areas can only be incorrectly assigned by a faulty program unit.
- UTM events initiated by KDCS calls are logged as failed if the assignment of KCRN, KCUS, or KCLT is incorrect. The incorrect assignment of KCRN, KCUS and KCLT can be caused not only by a faulty UTM program unit but also by a UTM user (e.g. if the user does not have authorization for the function called).
- From the SAT log records, you can identify whether an event was rendered ineffective by the subsequent rollback of the transaction:
- The log data record of the respective event contains the identification number of the transaction which initiated the event (log field UTMTAID).
- You can use the transaction identification to find the associated “end of transaction” log record (END-PU in the log field UTMSUBC). The transaction status is specified there in the UTMSTAT field.

The structure of the log records is described in the Appendix in the section “Structure of SAT log records” ("Structure of SAT log records").

- The KDCS call RSET (roll back the transaction while continuing the program unit run) implicitly initiates the “end of transaction” event with the transaction status “rollback” (OBJECT2= or UTMSTAT= R), and then initiates the “start of transaction” event (START-PU).

11.4 Postselection - evaluating log records

Data is logged by SAT in a highly protected global system file (SAT log file) under the BS2000 ID $SYSAUDIT.

With SATUT, data records can subsequently be selected from the SAT log file and output to a file (postselection). The log records can be selected using any logged field. A data record is only transferred if the data fields of the associated event fulfil certain rules. One of the following conditions can be defined for each field type:

- The value of the field matches an element of a list.
- The value of the field lies within certain limits.
- The data field of the specified field type is available in the data record.

For further information, see the BS2000 manual “SECOS”.

The structure of the log records is described in the Appendix "Structure of SAT log records".

11.5 Administration of SAT logging

SAT logs selected security-related UTM events for UTM applications. When generating the application, you define the processing result (success, failure) and the criteria (eventspecific, TAC-specific, or user-specific) on which SAT logging is to be based. This section describes the UTM SAT administration commands you can use to administer SAT logging for your UTM application.

The UTM SAT administration commands are separate transaction codes. They must therefore be defined when generating the application. The UTM SAT administrator can only call the UTM SAT administration functions using dialog TACs. The names of the transaction codes are listed in the following table:

Transaction code	Administration functions
KDCMSAT	Switch on and off SAT logging Change SAT logging values Query the command syntax
KDCISAT	Display information on SAT logging values Query the command syntax

In the UTM generation, you can define whether or not SAT logging is to be switched on automatically each time the application starts. If SAT logging is not switched on, you can nonetheless generate criteria for SAT logging, which can be switched on and off as required during operation.

The generated logging values can be changed using the administration command KDCMSAT. For event-driven logging, the changes apply only for the duration of the current application run. The changes are retained past the end of the application run for user and job-driven logging.

You can display the current values with the administration command KDCISAT.

UTM SAT administration commands are entered in line mode. Entries from output created with edit profiles and formatted entries will be rejected.

UTM SAT administration commands can only be called from UTM user IDs with UTM SAT administration authorization. UTM SAT administration authorization is assigned to a UTM user with USER ...,PERMIT=SATADM or PERMIT=(ADMIN,SATADM) when generating with KDCDEF. UTM administration alone (USER .., PERMIT=ADMIN) does **not** imply UTM SAT administration.

To be able to enter UTM SAT administration commands, the following conditions must be fulfilled when generating with KDCDEF:

- The administration program KDCSADM must be defined (PROGRAM statement). The sample program KDCSADM is supplied with openUTM.
- The UTM SAT administration commands KDCMSAT and KDCISAT must be defined as transaction codes (TAC statement with PROGRAM=KDCSADM and SATADM=Y).
- At least one user ID must be generated with UTM SAT administration authorization (USER statement with PERMIT=SATADM). Administration authorization can simultaneously be granted to several user IDs. UTM SAT administration authorization is linked to the user and not to a terminal, i.e. administration functions can be executed from any terminal. If administration functions are to be restricted to particular terminals, this is implemented with normal data access control functions (operands KSET= and LOCK=).

Example

The transaction codes for administration are defined with the TAC control statement, the administration program is defined with the PROGRAM control statement of the UTM tool KDCDEF:

```
PROGRAM KDCSADM,COMP=ILCS
:
TAC KDCMSAT ,PROGRAM=KDCSADM,SATADM=Y
TAC KDCISAT ,PROGRAM=KDCSADM,SATADM=Y
:
```

The operands PROGRAM=KDCSADM and SATADM=Y of the TAC statements can be omitted with the DEFAULT presetting:

```
DEFAULT TAC PROGRAM=KDCSADM,SATADM=Y
```

Notes

- Every access to the TAC KDCMSAT (apart from KDCMSAT HELP) is logged, even if SAT logging is switched off.
- UTM SAT administration using asynchronous jobs is **not** possible.

11.6 UTM SAT administration commands

- KDCISAT - query information on SAT logging values
- KDCMSAT - modify SAT logging

11.6.1 KDCISAT - query information on SAT logging values

The SAT administrator can use the KDCISAT command to obtain information on the values currently set for SAT logging or query the syntax of the KDCISAT command. KDCISAT indicates the UTM event classes from which the events are logged for a particular processing result (positive or negative). The information can also be queried for specific TACs and/or specific users. The UTM event classes are listed in the description of the EVENT operand in the KDCMSAT statement ("KDCMSAT - modify SAT logging").

The output for each KDCISAT inquiry indicates whether SAT logging is switched on or off.

`KDCISAT`	`{ [ TAC=tacname ] [, USER=username] | HELP }`

KDCISAT entered without operands:

> The output indicates the event classes from which events are logged. For the individual event classes, it specifies whether successful or failed events are logged (see Example 1). The values which are set identically for all users and for all TACs are output, i.e. all SAT logging values that were either generated in the SATSEL statement or were defined with the command KDCMSAT SATSEL=...,EVENT=(...).

TAC=tac

> From the individual event classes, it is indicated which events are specifically logged for this TAC (see Example 2).

USER=user

> From the individual event classes, it is indicated which events are specifically logged for this user.

TAC=tacname,USER=username

> For the individual event classes, it is indicated which events are logged in total for the specified user and the specified TAC. The output includes the defined event-specific values (KDCISAT without operand), the TAC-specific values (KDCISAT TAC=...), and the user-specific values (KDCISAT USER=...) for SAT logging (see Example 3).

HELP

> Indicates the syntax of this command.

Output of KDCISAT

All information is output at the terminal of the SAT administrator. The RESULT column indicates whether the system has logged events that indicate positive execution or negative processing. The value specified in RESULT is the result arising from the combination of the defined SAT logging conditions. The result is the inclusive-OR operation from the logging values defined for EVENT and possibly USER and possibly TAC (see the tables in section“Linking the preselection values”).

If an invalid USER or TAC is specified, then the UTM message `invalid TAC` or `invalid USER` is output.

Example 1

If you enter:

```
KDCISAT
```

the following is output:

SAT	USER	TAC	EVENT [1]	RESULT
ON			SIGN CHANGE-PW START-PU END-PU GSSB TLS ULS ADM-CMD	BOTH BOTH NONE NONE BOTH BOTH BOTH BOTH

[1]The meaning of the event classes (EVENT) and the values of RESULT are described for the KDCMSAT command (KDCMSAT - modify SAT logging).

Example 2

If you enter:

```
KDCISAT TAC=tac1
```

the following is output:

SAT	USER	TAC	EVENT	RESULT
ON		tac1	SIGN CHANGE-PW START-PU END-PU GSSB TLS ULS ADM-CMD	BOTH BOTH NONE NONE BOTH BOTH BOTH BOTH

Example 3

If you enter:

```
KDCISAT TAC=tac2,USER=user2
```

the following is output:

SAT	USER	TAC	EVENT	RESULT
ON	user2	tac2	SIGN CHANGE-PW START-PU END-PU GSSB TLS ULS ADM-CMD	BOTH BOTH NONE NONE BOTH BOTH BOTH BOTH

11.6.2 KDCMSAT - modify SAT logging

The conditions for the SAT audit are defined in the UTM generation. You specify the event classes from which events are to be logged, and you define the processing results (successful or failed execution) for user-specific, TAC-specific, or event-specific logging. The UTM SAT administrator can modify these generated values using KDCMSAT:

- Define the logging values with a separate statement for EVENT, TAC or USER.
- Activate or deactivate logging.

  In the UTM generation, you can preset the SAT logging values without activating SAT logging itself. When required, you can then switch on logging during operation with KDCMSAT SAT=ON. Logging can also be switched on if no values were preset in the UTM generation. In this case, you can specify the logging values with KDCMSAT.
- Query the syntax of this command.

The modifications for event-driven logging (SATSEL ...,EVENT=) and the switching on and off (SAT=ON/OFF) only apply until the end of the application run. Modifications for user and job-driven logging (SATSEL ...,TAC= and SATSEL...,USER=) are retained past the end of the application run, even for UTM-F. These values can be transferred to a new UTM generation with an inverse KDCDEF.

KDCMSAT	{ SATSEL={ BOTH | SUCC | FAIL | NONE | OFF }, { EVENT=(event1, ..., event *n*) | TAC=(tacname1, ..., tacname *n*) | USER=(username1, ..., username *n*) } *or:* SAT={ OFF | ON } *or:* HELP }

SATSEL =

Controls the type of SAT logging.

The value of SATSEL changes the generated logging type for the element of the application specified in the subsequent operand.

Each time you issue the KDCMSAT command, you can change the setting for one of the criteria EVENT, TAC, or USER.

BOTH	Both successful and failed events are logged.
SUCC	Successful events are logged.
FAIL	Failed events are logged.
NONE	No EVENT-specific, TAC-specific, or USER-specific event selection.

OFF — This setting is only possible for the EVENT criterion. In this case, no events are logged for the event classes specified in EVENT, even if SAT logging was activated in the UTM generation in the USER or TAC command.

EVENT=(event1, ..., event*n*)

Specifies the list of event classes for which the SAT logging conditions are to be changed. The following event classes can be specified and combined in any way. A maximum of 8 values can be specified.

SIGN — Sign-on of a user.

CHANGE-**P**W — Modification of the password by the user or the UTM administator.

START-PU — Start of a program unit run or acceptance of a dialog or asynchronous job.

END-PU — End of a program unit run.

GSSB — Access to a global secondary storage area.

TLS — Access to a terminal-specific long-term storage area (TLS).

ULS — Access to a user-specific long-term storage area (ULS).

ADM-CMD — Execution of an administration call.

TAC=(tacname1, ..., tacname*n*)

Specifies the list of transaction codes for which the conditions of SAT logging are to be changed. A list containing a maximum of 10 transaction codes can be specified.

USER=(username1, ..., username*n*)

Specifies the list of users (USER) for whom the conditions of SAT logging are to be changed. You can specify a list containing a maximum of 10 user names.

SAT=ON/OFF

Switch on/off SAT logging.

ON — SAT logging is switched on.

When SAT=ON, you can activate SAT logging when required during the application run for values that were preset in the UTM generation or were set with a preceding KDCMSAT command.

OFF SAT logging is switched off.

HELP

Shows the syntax of this command.

Output of KDCMSAT

The old and new SAT logging values, which were modified or switched on/off with this command, are output at the SAT administrator terminal.

Example 1

If you enter:

```
KDCMSAT SATSEL=SUCC,EVENT=(START-PU,END-PU)
```

the following is output:

SAT	EVENT	SATSEL	
		NEW	OLD
ON	START-PU END-PU	SUCC SUCC	FAIL FAIL

A corresponding table is output for the criteria TAC and USER.

Example 2

If you enter:

```
KDCMSAT SAT=ON
```

the following is output:

SAT	NEW	OLD
	ON	OFF

12 Accounting

openUTM provides accounting functions that enable the user of a UTM application to calculate the resources utilized by the users of a UTM application. The UTM accounting facility uses the resources of the operating system to determine the account data and enter the data in the operating system’s accounting file.

The data may be evaluated by the accounting procedure of the BS2000 system (RAV).

Detailed information on the BS2000 accounting system can be found in the BS2000 manual “RAV - Computer Center Accounting Procedure”.

The accounting functions that the corresponding operating system provides can only record the resource utilization and performance of a UTM application as a whole. However, if you want to be able to assign the computer resources used to individual users and charge the individuals accordingly, then the following must be taken into account for UTM accounting:

- The users of a UTM application are represented by the user IDs defined in the UTM generation and not by the user IDs of the operating system. You must therefore be able to assign the resources used by a user to individual UTM user IDs.
- A group of homogenous processes is active in a UTM application. Every process handles a series of jobs in succession for various users. The resources used within a process must therefore be determined for each service called (i.e. for individual program unit runs).
- The time conditions of OLTP operation require that the services be recorded in such a way that the performance of the application is not impeded.

UTM accounting therefore records the utilization of resources by the individual program units. This means that the resource utilization can be assigned to the transaction code (TAC) of the respective program unit and therefore to the UTM user who started the corresponding service.

In addition to the utilization of resources determined by UTM accounting, there is also a basic resource requirement that arises when a UTM application is running but which cannot be assigned directly to a user. These are:

- Disk space assignment for KDCFILE, SYSLOG, and USLOG files
- CPU utilization and I/Os for
  - starting and terminating UTM processes
  - handling connections for terminals
  - LPUT handling (transfer to USLOG file)
  - processing printer output

If the usage of these resources is to be taken into account, then you must charge these services at a flat rate to the users.

12.1 Definition of terms

This section provides a more detailed explanation of some of the terms that are relevant to UTM accounting.

Users in the sense of UTM accounting

The user of a UTM application for whom an account is to be created, is represented by the UTM user ID.

openUTM assigns the utilized resources to the LTERM partners as an alternative in UTM applications without real user IDs. The LTERM name of the connection user ID (TS applications and UPIC clients), the LU6 session name (LU6 partners) or the OSI association name (OSI TP partner) is used for applications or clients that have not explicitly signed on with a user ID.

In UTM applications without user IDs, openUTM assigns the resources used by terminals, UPIC clients or TS applications to the LTERM partners instead.

Accounting file

All information that the UTM accounting collects for the user-specific accounting of resources used is written by openUTM in the accounting file of the operating system.

The accounting file is administered by the BS2000 system administrator. The system administrator can evaluate this accounting file with the RAV tool.

Resources

This includes the following services:

- Technical DP services, particularly CPU utilization and I/Os
- Calling a particular program (program charge)

Calculation phase

The calculation phase is used as a starting point for the utilization of the accounting procedure.

In the calculation phase, openUTM determines the utilization of each resource for each program unit called and writes the values in the BS2000 accounting file as a calculation record. See section “Calculation phase” for more detailed information.

Calculation record

A calculation record is a record which openUTM writes in the BS2000 accounting file for each program unit run in the calculation phase. The accounting record type is UTMK. The data fields of the calculation record UTMK are described in the Appendix on "Structure of a calculation record".

Weight

A weight (factor) can be defined for each resource. This weight specifies how the resource is to be evaluated compared with other resources. The utilization of a resource is then introduced into the accounting procedure as the product “weight * resource utilization“. The weights for the individual resources are entered in the KDCDEF generation in ACCOUNT, see section “Determining the variant of the accounting procedure”.

Accounting phase

openUTM determines the resource utilization for each program unit. When the program unit terminates, openUTM calculates the sum of utilization values based on the weights and the generated fixed prices.

The following resources are taken into account:

- CPU utilization
- Input/output to disk
- Generated output jobs for printers
- Fixed price for calling a program unit

The result is a number of derived accounting units that are added to the user-specific accounting unit counter.

openUTM only then writes a record with the contents of this counter in the accounting file

- when the user signs off and is not signed on again to the UTM application via any other connection,
- when the application is terminated normally,
- or when a particular (generatable) maximum value is exceeded. You specify this maximum value in the KDCDEF generation with ACCOUNT ...,MAXUNIT= .

You must incorporate the weights in the generation of the application before the start of the accounting phase. You can choose between the following:

- Fixed-price accounting
- Utilization-oriented accounting
- Combination of both variants

You will find a detailed description of the accounting phase in section “Accounting phase”.

The accounting phase of UTM accounting can be enabled and disabled while the UTM application is running.

Accounting record

An accounting record is a record which openUTM writes to the accounting file in the accounting phase. The accounting record type is UTMA.
The data fields of the accounting record UTMA are described in the Appendix on "Structure of an accounting record"
.

Accounting units

Accounting units are the product of the utilization and weight of the respective resource. Only accounting units are counted in the UTM accounting facility. Using RAV, these units can be converted to costs, which are charged to the users.

Accounting unit counter

In a UTM application, openUTM keeps an accounting unit counter for each user and thereby accumulates the utilization of accounting units per user.

Fixed-price accounting

With this variant of the accounting function, a constant number of accounting units is calculated for a program unit run. This number is assigned to the transaction code when the application is generated. The weights of other resources are zero. In this manner you can also offer free services, e.g. informational functions.

Utilization-oriented accounting

With this variant of the accounting function, the current utilization of resources is calculated for a program unit run. The utilization values for the resource are weighted according to the generated weights. No fixed price is charged for calling program units.

Computer center accounting procedure (RAV)

The records written by UTM applications in the BS2000 accounting file can be processed further with RAV. The component RAV-UTM is available in RAV for this purpose.

12.2 Accounting phases

The following steps are required to execute accounting in UTM applications:

- Calculation phase
- Determination of the accounting procedure
- Accounting phase
- Evaluation

12.2.1 Calculation phase

The calculation phase provides approximate values that you can use to determine the weights and fixed prices for the utilization of a service. openUTM determines the resource utilization for each program unit run, creates a calculation record of type UTMK at the end of the program run and writes this record in the accounting file.

The calculation phase can also be enabled or disabled at any time via the UTM administration during live operation to check the generated weights and possibly to update them when regenerating, for example.

You should note, however, that openUTM writes a record in the accounting file after every program unit run when the calculation phase is activated. This has a negative impact on the performance of the application.

Activating the calculation phase

The calculation phase can be activated during KDCDEF generation or by administration, see openUTM manual “Generating Applications” and openUTM manual “Administering Applications”:

- KDCDEF statement ACCOUNT ACC=CALC
- or via UTM administration:
  - Using the KDCAPPL CALC=ON command
  - Or using WinAdmin/WebAdmin
  - Or using the KDCADMI program call KC_MODIFY_OBJECT with obj_type=KC_DIAG_AND_ACCOUNT_PAR

In BS2000 accounting the system administrator must activate the record type UTMK.

Deactivating the calculation phase

The calculation phase can only be deactivated by UTM administration:

- Using the KDCAPPL CALC=OFF command
- Using WinAdmin/WebAdmin
- Using the KDCADMI program call KC_MODIFY_OBJECT with obj_type=KC_DIAG_AND_ACCOUNT_PAR

Data of a calculation record

A calculation record contains the following data:

- Time stamp of BS2000 accounting
- Name of the UTM application
- Transaction code (TAC) of the program unit
- CPU utilization in the UTM task (msec)
- CPU utilization in the DB system (msec) if the DB system used returns the corresponding data to openUTM.
- Number of I/Os in the UTM task
- Number of I/Os in the DB system insofar as the database system provides the relevant data
- Length of the input message in bytes
- Length of the output message in bytes
- Number of output jobs to printers

- Accounting units for LTAC calls
- UTM users that have called the service
- Name of the LTERM partner through which the user is signed on
- Real time of the program unit run (msec)

Output messages that are intended for a follow-up program unit (e.g. after PEND ER) are also counted.

With RAV, the calculation records can be used to produce an evaluation which indicates the average resource utilization per TAC. If several UTM applications are running, an evaluation is produced for each UTM application.

12.2.2 Determining the variant of the accounting procedure

You must first determine if you want to use fixed prices, the utilization or a combination of these two variants for accounting purposes. Your decision depends on if you want to offer certain services of the application at fixed prices or if you want to charge for the actual resource utilization.

Fixed-price accounting

In fixed-price accounting, a program unit run costs a constant number of accounting units. These values are based on the values determined in the calculation phase. This makes fixed-price accounting the simplest solution.

You specify the number of accounting units in the KDCDEF generation in the TAC statement in the TACUNIT operand, see the openUTM manual “Generating Applications”.

```
TAC tacname,PROGRAM=progname,TACUNIT=number_of_accounting_units
```

The value specified in TACUNIT is added to the user-specific accounting unit counter for every transaction code called by the user.

You can also provide some services (e.g. informational functions) free of charge when using fixed-price accounting. You must generate the corresponding transaction codes as follows to do this:

```
TAC ... TACUNIT=0
```

With distributed processing, the same applies to the LTAC statement and the LTACUNIT operand, see section “Accounting with distributed processing”.

You must set the weights for the resources to 0 (default value) in the KDCDEF statement ACCOUNT when using fixed-price accounting.

Utilization-based accounting

In this variant the user is charged for the utilization of resources that are determined in the current accounting phase. You must specify weights for the individual resources. A weight is a factor that is multiplied with the number of units used. You can use the utilization data that you received in the calculation phase to help you choose the weights.

The weights are defined for each application in the KDCDEF statement ACCOUNT, i.e. they are valid for all program unit runs.

The determination of the weights is inevitably subjective and depends on the installation environment. You can assign weights to the following resources:

- CPU utilization (ACCOUNT operand CPUUNIT)
- I/O to background memory (ACCOUNT operand IOUNIT)
- Printer output (ACCOUNT operand OUTUNIT)

More details can be found in the openUTM manual “Generating Applications”.

Example for the UTM generation of this variant

```
ACCOUNT ACC=ON,CPUUNIT=15,IOUNIT=5,OUTUNIT=20
TAC tacname,PROGRAM=progname,TACUNIT=0
TAC ....
```

The following sum is then added to the accounting unit counter of the user for each transaction code call:

15 * CPU utilization + 5 * I/O utilization + 20 * printer output utilization

Combination of fixed-price and utilization-based accounting

You can also combine the two variants above for your accounting purposes by specifying a certain fixed price for calling a transaction code and then also charging for the utilization of resources (e.g. the CPU utilization).

The following sum is created and added to the accounting unit counter of the user in the accounting phase when a transaction code is called:

TACUNIT (fixed price for calling a program unit)
+ CPUUNIT * CPU utilization + IOUNIT* I/O utilization
+ OUTUNIT * printer output utilization

Example for the UTM generation of this variant

```
ACCOUNT ACC=ON,CPUUNIT=15
TAC tacnam1,PROGRAM=progname1,TACUNIT=1
TAC tacnam2,PROGRAM=progname2,TACUNIT=2
:
:
```

12.2.3 Accounting phase

In the accounting phase, openUTM determines the resources utilized per program unit run, calculates a weighted total from this figure and from the generated weights and fixed prices. openUTM then adds this result to the accounting unit counter of the UTM user. The value of this counter is contained in the accounting record which openUTM writes in the accounting file.

openUTM always writes an accounting record when a certain number of accounting units have been accumulated for the user, or when the user signs off and is not signed on to the UTM application via any other connection. The number of accounting units for which openUTM writes an accounting record is specified in the KDCDEF generation in ACCOUNT MAXUNIT=. You must note the following:

- You should not select a value for MAXUNIT that is too small because writing accounting records too often could affect the performance of the application negatively.
- You should not select a value for MAXUNIT that is too large because the accounting units that have not yet been written to the accounting file could be lost when the application crashes (accounting is not subject to transaction management).

After the accounting record has been written to the accounting file, the accounting unit counter and the counter for the number of TACs called are reset to zero.

Activating the accounting phase

With the KDCDEF control statement ACCOUNT ACC=ON, accounting is also activated for the UTM application in the UTM generation.

The accounting phase can also be activated and deactivated during live operation by the UTM administration:

- Using the KDCAPPL ACCOUNT=ON command
- Using WinAdmin/WebAdmin
- Using the KDCADMI program call KC_MODIFY_OBJECT with obj_type=KC_DIAG_AND_ACCOUNT_PAR

In BS2000 accounting, the system administrator must activate the record type UTMA.

Deactivating the accounting phase

The accounting phase can only be deactivated by administration:

- Using the KDCAPPL ACCOUNT=OFF command
- Using WinAdmin/WebAdmin
- Using the KDCADMI program call KC_MODIFY_OBJECT with obj_type=KC_DIAG_AND_ACCOUNT_PAR

Data of the accounting record

The accounting record is of record type UTMA. The accounting record contains the following data:

- Time stamp of BS2000 accounting
- Name of the UTM application
- UTM user ID
- Time the user signs on via the current connection
- Value of the accounting unit counter
- Number of TACs called with TACUNIT > 0 since the sign-on or since the last record was written

You can also collect calculation data while the accounting phase is running. This allows you to check the weights at any time.

12.2.4 Evaluation

The accounting records in the BS2000 system‘s accounting file are the result of the accounting phase. These records can be evaluated with RAV. The programs required here are components of RAV and are supplied with this product. See also the manual “RAV (BS2000/OSD) - Computer Center Accounting Procedure”.

The structure of the UTM accounting records is described in the Appendix on "Structure of the accounting records of openUTM".

12.2.5 Error situations

If BS2000 accounting cannot write an accounting record due to an error, e.g. because there is not enough space on the disk, openUTM generates message K079 and terminates the calculation and/or accounting phase. An insert of message K079 contains the cause of the error. The application continues execution.

After the error has been corrected, the calculation and/or accounting phase can be reactivated again by the UTM administration (e.g. using the administration command KDCAPPL).

12.3 Accounting with distributed processing

During distributed processing, every participating application can, in principle, start services in other applications. Accounting in distributed processing is primarily of use when the roles are unevenly distributed, i.e. one application acts entirely as the job submitter and other applications assume the job receiver roles. Consequently, in this section, the applications are referred to as **job-submitting applications** and **job-receiving applications**.

The job submitter application (job submitter) uses services provided by program units in remote partner applications (job receivers). In this case, the job-submitting application can be charged with the incurred resource utilization as a fixed price. Accounting units are assigned as a fixed price to the LTACs in the job-submitting application to do this. LTACs are the transaction codes that are defined in the job-submitting application for a service in a job-receiving application.

More details can be found in the openUTM manual “Generating Applications”, LTAC statement, LTACUNIT operand.

Calculation phase (determining the fixed price)

The average resource utilization of the program units that provided services for the job-submitting application is determined in the calculation phase in the **job-receiving appli-cation**. You can specify fixed prices based on the utilization values determined that will be charged to the users of LTACs in the job-submitting application.

openUTM counts the accounting units used in the LTAC calls in a field of the calculation record in the **job-submitting application**.

Accounting phase

In the **job-receiving application**, all utilization values that are incurred while processing jobs for a job-submitting application are assigned as follows:

- With LU6.1, to the sessions (LSES) to the job submitter
- With OSI TP, to the associations (OSI-LPAP ... ,ASSOCIATION-NAME=), if the OSI TP-job submitter did not sign on under a real user ID

The total for the services provided is therefore charged to the job-submitting application. The resources used by the individual users of the job-submitting application cannot be determined.

In the **job-submitting application**, openUTM adds the number of accounting units specified in the LTAC statement in the KDCDEF generation when an LTAC is called to the accounting unit counter of the user of the local application.

12.4 Restrictions

Please note the following when using UTM accounting:

- Transaction logging is not implemented when writing accounting information; this means that accounting units may be lost if an application crashes. The maximum value per user can be limited in the UTM generation.
- For applications with distributed processing, each LTAC call is counted in the calculation phase. No account is taken of whether or not a session could be opened following PEND processing.
- The recording of resource utilization begins before a program unit starts and ends with the processing of the PEND call. The remaining processing power (basic utilization) of the UTM tasks is not charged to the users.
- Rolling back a transaction has the following effects: All values except for CPU and I/Os are reset. Since openUTM accumulates the utilization values in the PEND processing, a rollback action can only reset utilization values if they originate in the current program unit run.
- If only asynchronous jobs have been processed for the user since the last application start, the sign-on time to the application is shown as zero in the accounting record.
- For the event exit VORGANG, the resource utilization is only recorded at the start of the service.
- For the event service BADTACS, the program unit weight cannot be taken into account in the accounting phase.

13 Checking performance with openSM2 and KDCMON

The performance of a UTM application is influenced by various factors. The determining factors lie on the one hand in the system environment of a UTM application (size of the working memory, performance capabilities of peripherals) and on the other hand in the UTM application itself (configuration of the application and structure of the program units). Performance checks should be carried out at regular intervals while an application is running, in order to detect performance bottlenecks at an early stage. The following tools are available for checking the performance of UTM applications:

- BS2000 Software Monitor openSM2
- UTM event monitor KDCMON with the evaluation tool KDCEVAL
- information services of UTM administration

You can also use the tools of the database system, e.g. SESCOS trace for SESAM/SQL.

Software Monitor openSM2

The Software Monitor openSM2 of the BS2000 system provides statistical data on performance and the utilization of resources. Together with the UTM-SM2 subsystem, it is also possible to determine and display application-specific data. These functions should be used during operation in order to monitor the behavior of a UTM application and identify performance bottlenecks.

UTM event monitor KDCMON

The UTM event monitor KDCMON is provided for UTM users. KDCMON records information on the runtime characteristics of UTM applications and user program units. If performance bottlenecks are detected, then you can collect data using **KDCMON**. You evaluate the data collected with the KDCEVAL tool. You can then carry out a detailed analysis based on this evaluation. See "Evaluation lists".

KDCMON is therefore an important tool for assessing the performance of a UTM application. KDCMON can be used to produce detailed performance evaluations when measurements with openSM2 or information of the UTM administration point to a performance bottleneck.

Information services in the UTM administration

Some information on evaluating the utilization of the application can also be queried using the **UTM administration information services**, e.g. via the KDCINF administration command or via the graphical administration tools WinAdmin/WebAdmin.

The KDCINF STATISTICS command provides, amongst others, data on the utilization of your UTM application. The KDCSINF STATISTICS command also allows you to obtain general statistical information on the application and obtain statistics for performance control as well as for assessing the performance of your UTM application during operation, for example application load, page pool utilization, number of users currently signed on, number of dialog or asynchronous transactions performed per second, open dialog and asynchronous services etc.

The KDCINF PAGEPOOL command supplies further, more detailed data on the current utilization of the page pool.

You can also use the administration command KDCINF SYSPARM to query whether or not the UTM application is supplying data to openSM2. For more information, see the openUTM manual “Administering Applications”.

If you administer the UTM application with the WinAdmin or WebAdmin graphical administration workstation, then you can also display the statistical data graphically.

13.1 Recording measurement data with openSM2

The Software Monitor openSM2 in BS2000 systems records statistical data on performance and the utilization of resources.

openUTM can supply openSM2 with UTM application data which is important for an initial evaluation of performance. The set of data is independent of the size of the configuration. The data indicates the behavior of the entire application.

The measurement data recorded by openSM2 is evaluated by the openSM2 component SM2R1, as well as the openSM2 component SM2-PA for evaluating user-specific measurement files. openSM2 collects measured values and, on request, outputs them directly to the terminal for realtime monitoring (online). openSM2 also collects the measurement data in a file, and the stored data can be evaluated at a later stage on request (offline).

Other openSM2 functions are also available for checking the performance of a UTM application, e.g. global process evaluation or measuring the processes of a UTM application and program analysis by SM2-PA.

Prerequisites for recording UTM measurement data by openSM2

To enable openUTM to supply data to openSM2 and to enable openSM2 to collect, store and edit UTM data, the following requirements must be fulfilled.

- The UTM-SM2 subsystem must be installed and loaded.

  In order for openUTM to supply data to openSM2, the UTM-SM2 subsystem is required. UTM-SM2 acts as a communication component between the tasks of the UTM application and openSM2. It is implemented as a separate subsystem and is included in BS2000-GA (basic configuration). For a description of how the BS2000 system administrator is to install the UTM-SM2 subsystem, see "UTM-SM2 subsystem" of the Appendix.

  UTM-SM2 can be loaded as follows:

  1. By the BS2000 system administrator with the command:

     ```
     /START-SUBSYSTEM SUBSYSTEM-NAME=UTM-SM2
     ```

  2. Automatically at the start of the application if the application is generated with MAX SM2=ON.
  3. When activating the supply of data via the administration (see below). The application must be generated with MAX SM2=OFF or MAX SM2=ON.

  If required, the UTM-SM2 subsystem can be unloaded during operation using the STOP-SUBSYSTEM command. This is necessary when exchanging the subsystem, for example. openSM2 and the UTM applications then automatically terminate their cooperation with UTM-SM2.

  After the subsystem has been loaded dynamically, data supply must be reactivated explicitly for each UTM application.

- Data supply from openUTM to openSM2 must be generated in the UTM application.

  In order that openUTM can supply UTM application data to openSM2, this function must be specified when generating the application. In this case, specify the value ON or OFF in the SM2 operand of the MAX statement.

  If you specify MAX...,SM2=ON, data supply to openSM2 is automatically activated when the application starts. If required, it can then be deactivated and reactivated again during operation using UTM administration functions.

  If you specify MAX...,SM2=OFF, data supply to openSM2 is permitted for this application. However, it must be explicitly activated during operation using UTM administration functions.

  If MAX ...,SM2=NO is generated, UTM does not supply data to openSM2 for this application. In this case, data supply cannot be activated by UTM administration either.

- Data supply to openSM2 is activated by UTM administration functions.

  The UTM administrator can use the command KDCAPPL SM2=ON to activate data supply to openSM2 if this was allowed for in the UTM generation. Data supply is deactivated with KDCAPPL SM2=OFF.

  Using the KDCINF SYSPARM command, the UTM administrator can define whether or not the application is permitted to supply data to openSM2 and whether it is currently supplying data.

  Data supply to openSM2 can also be activated and deactivated using the “program interface for administration” or using the graphical administration tools WinAdmin/WebAdmin.
- The openSM2 administrator (privileged openSM2 user) must initiate the collection of UTM data by openSM2.

  The openSM2 administrator must use the command

  `START-MEASUREMENT-PROGRAM TYPE=UTM`

  to instruct openSM2 to collect and evaluate data on UTM applications. The STOP-MEASUREMENT-PROGRAM command is used to deactivate the function. If the UTM-SM2 subsystem is loaded, the collection of UTM data can be activated at any time on the openSM2 side, regardless of whether or not data supply is activated on the UTM side. However, openSM2 cannot begin to process the data until data supply is activated by openUTM.

Output and evaluation of measurement data

The openSM2 user can display the data supplied by openUTM on an openSM2 screen in online mode (UTM report or UTM application report). openSM2 also stores the data in a global system measurement file. This means that the data can also be evaluated by SM2R1 at a later stage.

The openSM2 screen “UTM-REPORT” contains a tabular overview of data for all UTM applications currently supplying data to openSM2. One line is output for each UTM application.

The openSM2 screen “UTM-REPORT” can be output periodically with the command:

```
REPORT UTM
```

Current measured values for one or more selected UTM applications are supplied in the openSM2 screen “UTM-APPLICATION-REPORT”.

The openSM2 screen “UTM-APPLICATION-REPORT” can be output periodically with the command:

```
SELECT-UTM-APPLICATION (application1,application2,...)
```

For *application1*,*application2*,... specify the names of the UTM applications whose behavior you want to monitor in online mode. For *application1,..* you must specify the application name generated in the MAX statement.

The meaning of the data output is explained in the “openSM2” User Guide; the terms used are consistent with openUTM usage.

In addition to the option of monitoring the measurement data online, the following evaluations of data from UTM applications are possible with SM2R1:

- reports 128 through 133
- SUMMARY UTM

See also the “openSM2” User Guide.

13.2 KDCMON - UTM event monitor

KDCMON is a functionally limited variant of COSMOS. COSMOS is a tool implemented in BS2000 for checking performance. Only UTM events and certain DB events are recorded with KDCMON. It is possible to run KDCMON and COSMOS simultaneously on a system; openSM2 and KDCMON can also be implemented together.

KDCMON can be activated during operation and then deactivated again after the desired monitoring period. The data can be written to tape or disk. With larger volumes of data, the data should be recorded on tapes; this avoids backlogs in the data entry.

The tools KDCPMSM and KDCEVAL are available for evaluating the data recorded by KDCMON:

- KDCPMSM converts the data recorded by KDCMON and sorts it
- KDCEVAL generates the evaluation lists from the converted data

KDCMON can also be implemented when openUTM versions are operated in parallel, i.e. KDCMON can record data from UTM applications running under various openUTM versions in the same BS2000 system.

13.2.1 Starting and stopping data recording

Data recording can be started in two steps:

- First you must start KDCMON.
- Then activate data recording for the UTM applications to be checked.

KDCMON is an independent subsystem in the BS2000 system and is included in BS2000 basic configuration. KDCMON must be installed and loaded by the system administrator. See also section “KDCMON subsystem”.

Starting KDCMON

Before starting KDCMON, you must create a file in which KDCMON is to write the recorded data. To do this, issue the following commands:

- For recording to disk files:

```
/SET-FILE-LINK LINK-NAME=KDCMON,FILE-NAME=kdcmonfile -
/     ,ACCESS-METHOD=UPAM
/MODIFY-FILE-ATTRIBUTES FILE-NAME=kdcmonfile        -
/     ,SUPPORT=*PUBLIC-DISK(SPACE=*RELATIVE         -
/      (PRIMARY-ALLOCATION=xxx,SECONDARY-ALLOCATION=yyy))
```

  xxx and *yyy* must be multiples of 12 (otherwise DMS error).

- For recording to tape files:

```
/SET-FILE-LINK LINK-NAME=KDCMON,FILE-NAME=kdcmonfile -
/     ,ACCESS-METHOD=BTAM
/MODIFY-FILE-ATTRIBUTES FILE-NAME=kdcmonfile  -
/ ,SUPPORT=*TAPE(VOLUME=xxxxxxx,DEVICE-TYPE=type)
```

> The block size and record length must not be specified for the file.

You can then start KDCMON under the ID $TSOS. The program for starting KDCMON is provided in the file SYSPRG.KDCMON.*nnn*:

```
/START-EXECUTABLE-PROGRAM FROM-FILE=$userid.SYSPRG.KDCMON.nnn
```

The version identifier *nnn* stands for the BS2000 version in which KDCMON is running. See also section “KDCMON subsystem”.

> You can also start KDCMON using the SDF command START-KDCMON, see section “Starting UTM tools via separate SDF commands”.

The KDCMON program expects the following control parameters:

BUFPAG=size

Specifies the size of the buffer in KDCMON in units of 4 KB.

Permitted values: 1 to 7
Default value: 2 (= recommended value)

BUFFER=number

Specifies the number of buffers in KDCMON.

Permitted values: 2 to 128
Default value: 4 (= recommended value)

{TIME= mmm | START | BREAK}

Determines the runtime of KDCMON.

The specification of one of TIME=*mmm*, START or BREAK terminates the input of the control parameters.

TIME=mmm Runtime of KDCMON in minutes. After the specified time has elapsed, KDCMON terminates.

Minimum value: 1
Maximum value: 150

START KDCMON runs for 30 minutes. Corresponds to the specification TIME=30.

Default value: START

BREAK KDCMON runs until RESUME is entered.

> **CAUTION!**
>
> KDCMON can run for a maximum of 150 minutes; otherwise, the evaluation tool KDCEVAL cannot evaluate the data.

The buffers have the same meaning in KDCMON as in COSMOS.

The program SYSPRG.KDCMON.*nnn* must not be terminated during data entry, as otherwise the KDCMON subsystem will no longer be available and must be unloaded.

Stopping KDCMON

KDCMON terminates after the predefined time set in the parameters TIME=*mmm* and START, or after RESUME is entered with BREAK. However, the KDCMON subsystem remains loaded and can be unloaded using the DSSM command

```
/STOP-SUBSYSTEM KDCMON
```

if data entry is not running at this time. If you want to unload KDCMON when data entry has not concluded in one of the UTM applications, you must specify:

```
/STOP-SUBSYSTEM KDCMON,FORCED=YES
```

Activating and deactivating data recording

After KDCMON has been started successfully, you can record data from all UTM applications running on the respective system.

KDCMON also records BCAM wait times: If KDCMON is activated for input messages, BCAM records the wait time spent by these messages in the BCAM transport system before they are picked up by openUTM. For connections with intense dialog, the wait times are recorded for all input messages. For other connections, a statistical selection is made in high-load situations. openUTM transfers this wait time for each message from BCAM and writes it in the KDCMON records. From this data, KDCEVAL determines the maximum, minimum, and mean value (specified in seconds). These values are output in the WAIT list (see "WAIT: WAITING TIMES").

Using UTM administration functions, you can define the UTM applications from which data is recorded by KDCMON. Data entry, including the entry of BCAM wait times, can be activated and deactivated using the administration command:

```
KDCDIAG KDCMON={ ON | OFF }
```

This administration function is also available on the KDCADMI program interface and via the graphical administration tools WinAdmin/WebAdmin.

You can activate and deactivate data entry for an application during a KDCMON run several times. Up to 10 data acquisition time intervals are possible.

The UTM administrator can use the following command:

```
KDCINF SYSPARM
```

at any time to determine whether or not data is being recorded.

If openUTM detects that the KDCMON function is not available when it attempts to activate it, then the following message is output to the default destination SYSLOG:

```
K080 KDCMON is not active
```

Possible cause:
The KDCMON subsystem was not started or was not installed under $TSOS.

If openUTM detects that the KDCMON function is not available any more while it is acquiring data, then openUTM deactivates the collection of data and informs the user of this fact with message K080.

openUTM logs the activation and deactivation of the BCAM wait time with UTM message K146; the default destination of the UTM message is SYSLOG. UTM message K146 is also output by openUTM if an error occurs when reading the BCAM wait times. As diagnostic documentation, openUTM creates a UTM dump with the dump error code ASIS70, UMES02, or WAIT61. The application then continues to run without recording the BCAM wait time.

13.2.2 Evaluating data

In order to evaluate the KDCMON data, the log files must first be converted from PAM or BTAM format to SAM format using the KDCPMSM tool.
The records must then be sorted on the basis of the time stamp recorded in each record. This is necessary because it is not guaranteed that the records are sorted in chronological order in the file.

13.2.2.1 Converting the data to the SAM format and sorting the data

The PAMSAM procedure is made available to the user in the procedure library SYSPRC.UTM.070 for converting and sorting the data.

Before calling the procedure, you must assign the KDCMON file:

- Disk files

  /SET-FILE-LINK LINK-NAME=KDCMON,FILE-NAME=kdcmonfile,ACCESS-METHOD=UPAM
- Tape files

```
/SET-FILE-LINK LINK-NAME=KDCMON,FILE-NAME=kdcmonfile,ACCESS-METHOD=BTAM
/MODIFY-FILE-ATTRIBUTES FILE-NAME=kdcmonfile -
/      ,SUPPORT=*TAPE(VOLUME=xxxxxxx,DEVICE-TYPE=TAPE-C4)
```

The PAMSAM procedure is called as follows:

```
/CALL-PROCEDURE NAME=SYSPRC.UTM. 070(PAMSAM),PROCEDURE-PARAMETERS=( -
/                              [,KDCPMSM=kdcpmsm-progname][,SAMFILE=samfile] -
/                               [,SORT=sortprogram])
```

Meaning of the parameters and default values

kdcmonfile PAM or BTAM file with the recorded data to be converted.

kdcpmsm-progname

Name of the KDCPMSM tool in the file SYSPRG.KDCMON.*nnn*.KDCPMSM:
nnn=190 for BS2000 OSD/BC V10.0
nnn=200 for BS2000 OSD/BC V11.0

samfile Name of the output file in SAM format in which PAMSAM is to write the data records sorted according to time stamp.

sortprogram Name of the BS2000 utility SORT.

The file `KDCMON.WORK` is created while the procedure is running; this file is deleted again after the sorting run.

Following the procedure run, the data converted to SAM format and sorted according to time stamp is contained in the file *samfile*, which was specified in the SAMFILE parameter.

If the KDCPMSM tool is terminated incorrectly, process switch 3 is set to ‘on’.

13.2.2.2 Evaluating data with the KDCEVAL tool

The sorted data can be evaluated with KDCEVAL. The evaluation can be carried out interactively and in batch mode. KDCEVAL requires you to enter parameters for control purposes.

Only data from **one** application can be evaluated in a run. The user can restrict the evaluation to part of the recorded data by specifying a desired time interval (parameter TIME=). If data entry was activated and deactivated several times for the application within the evaluation time limits, a separate evaluation is carried out for each entry period (from KDCMON=ON to KDCMON=OFF), whereby a maximum of 10 such entry periods are possible.

The evaluation tool KDCEVAL is started with:

```
/SET-FILE-LINK LINK-NAME=KDCMON,FILE-NAME=samfile                    -
/                    ,BUFFER-LENGTH=BY-CATALOG                       -
/                    ,BLOCK-CONTROL-INFO=BY-CATALOG
/ START-EXECUTABLE-PROGRAM                                           -
/ FROM-FILE = *LIBRARY-ELEMENT(                                      -
/                      LIBRARY = SYSLNK.UTM. .UTIL                   -
/                     ,ELEMENT-OR-SYMBOL = KDCEVAL                   -
/                     ,TYPE = L )
```

070

> You can also start KDCEVAL using the SDF command START-KDCEVAL, see section “Starting UTM tools via separate SDF commands”.

Meaning of parameter

samfile

Name of the output file of PAMSAM, i.e. the SAM file with the sorted data records.

After the evaluation program has been started interactively, KDCEVAL outputs the following message to request the input of control parameters:

```
PLEASE ENTER COMMANDS OR 'HELP' OR 'END'
```

KDCEVAL control parameters

The program reads the SYSDTA parameters from SYSDTA. The individual commands you can use to control the evaluation have the following format:

APPLINAME applicationname

Name of the application for which the evaluation is to be carried out. If the file contains data from several applications, a separate evaluation must be implemented for each application. It is not possible to carry out an evaluation for more than one application during one KDCEVAL run.

TIME FROM={ t1 | START }, TO={ t2 | STOP }

Time specification for defining the evaluation time limits.

FROM=t1

Start time of the evaluation in seconds.
The time is specified relative to the start of KDCMON in seconds.

FROM=START

The evaluation starts at the beginning of the file.

TO=t2

End time of the evaluation.
The time is specified relative to the start of KDCMON in seconds.

TO=STOP

The evaluation continues until the end of the file.
The following apply for *t1* and *t2*:
Minimum value: 0
Maximum value: 99999999

LIST { ($list_1$, $list_2$,...,$list_n$ [,TABLE]) | (STD [,TABLE]) | (ALL [,TABLE]) }

$list_1$, $list_2$,...,$list_n$

Names of the individual lists to be prepared. The names that you can specify here are indicated on "Evaluation lists". The TRACE and TRACE2 lists must not be specified at the same time.

STD

This evaluation covers the lists TASKS, SUMM, TIMES and TCLASS.

ALL

The evaluation covers all lists apart from TRACE and TRACE2.
If TRACE or TRACE2 is also to be evaluated, you must specify LIST (ALL, TRACE) or LIST (ALL, TRACE2).
If ALL or STD is specified without TABLE, the round brackets can be omitted.

TABLE

If TABLE is specified in addition, the lists are created in a table format that can be further processed on PC with Excel or another spreadsheet program, see "Processing evaluation data on the PC". TABLE only works on the segregated lists TASKS, TIMES, TCLASS, TACCL, TACPT and TACLIST.

OPTION **D**ECIMAL-**S**EPARATOR={ **C**OMMA | **P**OINT }, SHOW-TSN={ **F**IRST | **A**LL }

DECIMAL-SEPARATOR=COMMA

The comma is used as the decimal separator.

DECIMAL-SEPARATOR=POINT

The period is used as the decimal separator; this is the default value.

SHOW-TSN=FIRST

In the TRACE2 list, in successive records with identical TSN, the TSN is only output in the first record in the sequence. The subsequent records in this kind of sequence of records with identical ID contain quotation marks (") in the TSN field.
This is the default value.

SHOW-TSN=ALL

In the TRACE2 list, the TSN is output in each record. This can be useful if the output list is to be edited or analyzed using a program.

END

This command terminates parameter input.

The HELP command can also be entered with interactive evaluations. The syntax of the commands and the possible list names are output in this case.

Errors and messages

- If KDCEVAL is terminated incorrectly, process switch 3 is set to ‘on’.
- If one of the commands APPLINAME, TIME or LIST is missing, the evaluation is aborted with the following error message:

  `MANDATORY COMMAND MISSING`
- In the case of a syntax error, the following message and the incorrect command are displayed:

  `ERROR IN COMMAND`
- If the time specifications *t1* and *t2* are inconsistent, the following message is output:

  `KDCEVAL: WRONG TIME INPUT`
- If no records are found in the file for the application or if no data exists within the evaluation time limits, one of the following messages is output:

  `NO EVALUATION : NO RECORD WITH APPLINAME FOUND`

  or

  `NO EVALUATION : NO RECORD IN TIME_INTERVAL`
- If a DMS error occurs, the following messages are output:

  `KDCEVAL: DMS-ERROR Dxxx FOR INPUT FILE`

  `KDCEVAL: NO EVALUATION`

  (Dxxx = DMS error code)
- Version check:

  It is only possible to evaluate KDCMON data using KDCEVAL if KDCEVAL has the same openUTM version as the UTM system code. KDCEVAL checks the version of the KDCMON data. If KDCEVAL identifies an illegal version, KDCEVAL aborts the evaluation with the following message:

  `NO EVALUATION: INPUT FILE FROM INVALID UTM VERSION`

Result of the KDCEVAL evaluation

KDCEVAL writes the result of the evaluation into the files of a file generation group (FGG) with the name:

`KDCMON.appliname`

The FGG contains a maximum of 10 generations. When the evaluation is started, any existing file generation of this name is deleted.

If data entry was deactivated several times during a KDCMON run, then KDCEVAL writes the evaluation data of each interval to a separate file of the FGG.

13.2.3 Processing evaluation data on the PC

If you specify the TABLE operand in addition to the list name in the LIST control parameter for KDCEVAL, the lists are created in table form. This type of processing is only possible for TASKS, TIMES, TCLASS, TACCL, TACPT, and TACLIST lists.

The lists generated in this way can be processed and formatted graphically on a PC using a spreadsheet program such as Microsoft Excel. The macro `kdceval.xls` is supplied on the WinAdmin data medium for Excel.

Carry out the following steps:

1. Transfer the list file generated by KDCEVAL to a PC using ftp or openFT and copy the macro `kdceval.xls` from the WinAdmin data medium on this PC.

   The macro requires that the file to be evaluated has the suffix `.txt`.
2. Call the macro (`kdceval.xls`) and read the list file into Excel. Excel then creates a separate spreadsheet for each list, as well as an additional sheet with summary information.
3. Process the individual lists as desired, e.g. by sorting a list and then converting it into a curve chart or bar chart.

13.2.4 Evaluation lists

Each evaluation list includes the following:

- a title containing the name of the evaluation list
- a header, which is identical for all lists
- the specific evaluation list

The list header is structured as follows:

NAME OF APPLICATION : applname DATE : yyyy-mm-dd

BS2000 VERSION :V190 openUTM VERSION: $(version0VDot)

COMMENCEMENT TIME : rel-sec SEC. KDCMON START : hh:mm:ss

KDCEVAL VERSION:V07.0A00

END TIME : rel-sec SEC. APPLICATION RECORDING START : hh:mm:ss

APPLICATION RECORDING END : hh:mm:ss

SYSTEM INFORMATION : system info ,Number CPUs : nn, Bitmode : nn Bit

APPLICATION INFO : Appli-Mode S Cache size,loc BLKSIZE : 4K Test-Mode OFF Data Compression OFF

The fields are explained below:

NAME OF APPLICATION	Name of the application.
DATE	Date of data entry with KDCMON.
BS2000 VERSION	BS2000 version of the system on which KDCMON was running.
COMMENCEMENT TIME	Start time of the selected evaluation period (relative to the start time of KDCMON)
KDCMON START	Start time of KDCMON.
END TIME	End time of the selected evaluation period (relative to the start time of KDCMON).
APPLICATION RECORDING START	Start time of data recording for the UTM application.
APPLICATION RECORDING END	End time of data recording for the UTM application.
Cache size, loc	Size and storage location of the cache. The following output is possible for storage location:
	PS if the cache is located in the program space.

	DS if the cache is located in the data space.

The processing times are always the ELAPSED TIME (real time).

The following individual evaluations and combinations of evaluations are possible:

TASKS	UTILIZATION OF THE UTM TASKS
SUMM	TRANSACTION EVALUATION
TIMES	DISTRIBUTION OF PROCESSING TIMES
KCOP	KDCS CALLS STATISTIC
WAIT	WAITING TIMES
TCLASS	EVALUATION OF THE TAC CLASSES
TACCL	TAC SPECIFIC TAC CLASS EVALUATION
TACPT	TAC SPECIFIC DISTRIBUTION OF PROCESSING TIMES
TACLIST	TAC SPECIFIC STATISTICS
TRACE	TASK SPECIFIC TRACES
TRACE2	TASK PERFORMANCE TRACES

The individual evaluation lists are described below.

13.2.4.1 TASKS: UTILIZATION OF THE UTM TASKS

This list provides an overview of the utilization levels of the processes of the application. Furthermore, the CPU utilization and the number of input and output operations (I/Os) are indicated for each individual UTM process and the sum is displayed for all tasks of the application.

```
                          1 = Program
                          2 = System code
                          3 = Database
                          4 = Bourse Wait
|------------------------|
| TSN  | START TIME      | TASK UTILIZATION    , Number Used Tasks:   6 , Number System Tasks:   0
| 3FYA | 16:46:32.928063 | <--------1-------><----2----><----3---><--------------4-------------->
| 3FYB | 16:46:32.917157 | <--------1-------><---2---><---3---><-----------------4---------------->
| 3FYC | 16:46:37.661807 | 2<---------------------------------4---------------------------------->
| 3FYD | 16:46:32.924186 | <---------------------------------4---------------------------------->
| 3FYE | 16:46:32.936194 | <--------1-------><---2---><----3---><-----------------4---------------->
| 3FYF | 16:46:32.914551 | <--------1-------><---2---><----3----><----------------4---------------->
|------|-----------------|
TSN      CPU-time  Number I/O     Program      System    Database  Bourse Wait  System Task
3FYA        20056        7150       70596       38931       35553       148124            N
3FYB        19133        6904       64621       35138       32592       160865            N
3FYC         1852         933          19        5009         111       283332            N
3FYD         1084         490          46        2808          29       290326            N
3FYE        19389        6964       66356       34615       35336       156890            N
3FYF        19188        6995       65410       34683       32640       160486            N
Summ        80702       29436
```

Explanation of the terms in the list:

TSN	TASK SEQUENCE NUMBER of the UTM process.
START TIME	Time of the first record of this process (absolute).
Program	Proportion of processing time of the application program in the UTM process. This time also includes the SVC handling of the SVCs called from the application programs.
System code	Proportion of processing time of the UTM system code.
Database	Proportion of time required to process the database calls. If the processing of a database call necessitates a process switch, this time also includes the wait time of the UTM process.
Bourse Wait	Proportion of time awaited by the process for new jobs to enter the job queue.
System Task	Specifies whether this task is a UTM system process for the application.

The times output in the columns Program, System, Database and Bourse Wait are real times. The unit used is milliseconds (in the same way as for the CPU time). A reduction in the number of tasks during the evaluation time limits must be avoided for the TASKS evaluation as this would lead to distorted results. For such a use case you should use separate evaluation time limits.

13.2.4.2 SUMM: TRANSACTION EVALUATION

This list provides an overview of the services and transactions for the evaluation period. The list only includes transactions that lie completely within the evaluation period. The evaluation tool KDCEVAL also indicates the CPU utilization of all program unit runs that were terminated within the evaluation time limits:

```
|-------------------------------------------------------------|
| COUNT OF TRANSACTIONS                          :       8229 |
|                                                             |
| COUNT OF SERVICES                              :       8229 |
|                                                             |
| COUNT OF DIALOG STEPS                          :       8229 |
|                                                             |
| NUMBER OF DIALOG STEPS PER SECOND              :      28.08 |
|                                                             |
| NUMBER OF DB CALLS PER TRANSACTION             :          9 |
|                                                             |
| TOTAL CPU-TIME USED IN MSEC                    :      77741 |  (1))
|-------------------------------------------------------------|
```

1)This line indicates the total CPU utilization of the program unit runs. This also includes the CPU utilization in the UTM and operating system code, insofar as this occurs within the program unit runs, as well as the start and end processing for the program units in openUTM. Other actions of the UTM tasks that do not belong directly to program unit runss are not included in these values.

13.2.4.3 TIMES: DISTRIBUTION OF PROCESSING TIMES

In tabular form, this list indicates a distribution of processing times in milliseconds for the program units. These times do not include the wait time before processing by openUTM.

The list has the following format:

PROCESSING TIMES (MSEC)	NUMBER	PERCENT
0 - 100	76	58.91
101 - 200	45	34.88
201 - 500	7	5.42
501 - 1000	1	0.77
1001 - 2000	0	0.00
2001 - 5000	0	0.00
5001 - 10000	0	0.00
10001 - 20000	0	0.00
20001 - 50000	0	0.00
50001 - 100000	0	0.00
> 100000	0	0.00

This list indicates the number of complete program unit runs and the percentage for the respective time class.

13.2.4.4 KCOP: UTM CALLS STATISTIC

This table specifies how often the individual UTM calls occurred in the evaluation period.

Calls that are not included in the list of calls known to KDCEVAL appear under *others*.

This list also contains calls that are issued by openUTM for internal processing and are not available to the user:

CONT	Call following formatting or database communication.
ADMI	UTM administration action
WAIT	End of processing of a program run.
NOOP	Record event data for DB calls.

The KCOP list has the following format:

OP	OM	NUMBER		OP	OM	NUMBER
ADMI		7		MPUT	HM	0
APRO	AM	0		MPUT	ID	0
APRO	DM	0		MPUT	NE	0
APRO	IN	0		MPUT	NT	34761
CONT		17338		MPUT	PM	0
CTRL	AB	0		MPUT	RM	0
CTRL	EC	0		NOOP		0
CTRL	PE	0		PADM	AC	0
CTRL	PR	0		PADM	AI	0
CTRL	SC	0		PADM	AT	0
DADM	CS	0		PADM	CA	0
DADM	DA	0		PADM	CS	0
DADM	DL	0		PADM	OK	0
DADM	MA	0		PADM	PI	0
DADM	MV	0		PADM	PR	0
DADM	RQ	0		PEND	ER	0
DADM	UI	0		PEND	FC	0
DGET	BF	0		PEND	FI	8231
DGET	BN	0		PEND	FR	0
DGET	FT	0		PEND	KP	0
DGET	NT	0		PEND	PA	0
DGET	PF	0		PEND	PR	0
DGET	PN	0		PEND	PS	0
DPUT	NE	0		PEND	RE	0
DPUT	NI	0		PEND	RS	0
DPUT	NT	0		PEND	SP	0
DPUT	RP	0		PGWT	CM	0

```
| DPUT  QE                0   |    | PGWT  KP                0   |
| DPUT  QI                0   |    | PGWT  PR                0   |
| DPUT  QT                0   |    | PGWT  RB                0   |
| DPUT  +I                0   |    | PGWT  RT                0   |
| DPUT  -I                0   |    | PGWT  ST                0   |
| DPUT  +T                0   |    | PTDA                    0   |
| DPUT  -T                0   |    | QCRE  NN                0   |
| FGET                    0   |    | QCRE  WN                0   |
| FPUT  NE                0   |    | QREL  RL                0   |
| FPUT  NT                0   |    | RSET                 6907   |
| FPUT  RP                0   |    | SGET  GB                0   |
| FPUT  UF                0   |    | SGET  KP                0   |
| GTDA                    0   |    | SGET  RL                0   |
| INFO  CD                0   |    | SGET  US                0   |
| INFO  CK                0   |    | SIGN  CK                0   |
| INFO  DT                0   |    | SIGN  CL                0   |
| INFO  FH                0   |    | SIGN  CP                0   |
| INFO  GN                0   |    | SIGN  OB                0   |
| INFO  LO                0   |    | SIGN  OF                0   |
| INFO  PC                0   |    | SIGN  ON                0   |
| INFO  SI                0   |    | SIGN  ST                0   |
| INIT                 8230   |    | SMSG                    0   |
| INIT  PU                0   |    | SPUT  DL                0   |
| INIT  MD                0   |    | SPUT  ES                0   |
| LPUT                    0   |    | SPUT  GB                0   |
| MCOM  BC                0   |    | SPUT  MS                0   |
| MCOM  EC                0   |    | SPUT  US                0   |
| MGET                 8230   |    | SREL  GB                0   |
| MGET  NT                0   |    | SREL  LB                0   |
| MPUT  CM                0   |    | UNLK  DA                0   |
| MPUT  EM                0   |    | UNLK  GB                0   |
| MPUT  ES                0   |    | UNLK  US                0   |
| MPUT  GC                0   |    | WAIT                 8232   |
| OTHERS                  0   |    |                             |
|-----------------------------|    |-----------------------------|
```

13.2.4.5 WAIT: WAITING TIMES

To establish bottleneck situations, openUTM inserts measuring jobs into the job queue at regular intervals if KDCMON is activated. The wait time of the jobs in the UTM queue can be determined on the basis of the time at which the job was introduced and the time of processing. The time difference between the individual pseudo jobs is approximately 10 seconds.

The following information is logged in the WAIT list:

- The WAITING TIME column indicates the established wait time for each pseudo job in seconds.
- For these wait times, the evaluation tool KDCEVAL also calculates the maximum, minimum, and mean value in seconds and outputs these values under UTM WAITING TIMES.
- The NUMBER OF TASKS column indicates the number of processes available in the application at this time. The UTM system processes are not included in this number.
- The BCAM wait times are output in seconds in the BCAM WAITING TIMES section. BCAM V12.0 or later measures the time spent in the BCAM transport system by input messages for the application before they are retrieved by openUTM.

  NUMBER OF MESSAGES indicates the number of messages for which BCAM supplied a wait time. When KDCMON recording is activated (and hence also time recording in BCAM), there are generally messages already in the BCAM message pool. When openUTM retrieves these messages from BCAM, openUTM does not receive a wait time for them. These messages are not included in the value for NUMBER OF MESSAGES.

If the wait time is too long, the number of UTM tasks should be increased.

The WAIT list has the following format:

TIME STAMP	WAITING TIME	NUMBER OF TASKS
10:33:35.742	0.018	1
10:33:45.781	0.008	2
10:33:55.841	0.007	3
10:33:66.027	0.008	3
10:33:76:087	0.008	3
10:33:86:112	0.007	3
10:33:96:123	0.007	3

```
                      UTM WAITING TIMES:
TIME STAMP :  10:33:35.742    WAITING TIME MAXIMUM :        0.018
TIME STAMP :  10:33:41.740    WAITING TIME MINIMUM :        0.007
NUMBER OF ENTRIES:       7    WAITING TIME AVERAGE :        0.009
                     BCAM WAITING TIMES:L
TIME STAMP :  10:23:35.742    WAITING TIME MAXIMUM :        0.728
TIME STAMP :  10:39:35.740    WAITING TIME MINIMUM :        0.027
NUMBER OF MESSAGES:    200    WAITING TIME AVERAGE :        0.125
```

13.2.4.6 TCLASS: EVALUATION OF THE TAC CLASSES

The TCLASS list contains an overview of job processing of TACs in the individual TAC classes (1 through 6) in tabular form. In the evaluation, all dialog TACs to which no TAC class was assigned during generation with KDCDEF are combined into TAC class 0.

In the UTM generation, the user can define the maximum number of tasks that can operate for a TAC class at any one time. When this number is reached, subsequent jobs are placed in a TAC class-specific queue.

TAC-CLASS	NUMBER CALLS	DISTRIBUTION IN PERCENT			AVERAGE WAIT TIME (IN MSEC)	MAXIMUM WAIT TIME (IN MSEC)	MINIMUM WAIT TIME (IN MSEC)
		NUMBER CALLS	WAIT TIME=0	WAIT TIME>0			
0	21	30.87					
1	2	2.94	100.00	0.00	0	0	0
2	4	5.88	75.00	25.00	3000	5000	1000
3	3	4.41	100.00	0.00	0	0	0
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
8	5	7.35	80.00	20.00	5000	8000	2000
9	2	2.94	90.00	10.00	4000	0	0
10	3	4.41	100.00	0.00	0	5000	1000
11	4	5.88	75.00	25.00	3000	0	0
12	3	4.41	100.00	0.00	0	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	8000	2000
16	5	7.35	80.00	20.00	5000		
			90.00	10.00	4000		

```
41 DIALOG TACS WERE CALLED

27 ASYNCHRONOUS TACS WERE CALLED
```

The TCLASS list contains the following information:

- The NUMBER OF CALLS column indicates the number of TAC calls in the evaluation period for a TAC class.
- The DISTRIBUTION IN PERCENT column contains percentage values. The subcolumn NUMBER CALLS specifies the percentage of calls of a TAC class within the number of all TAC calls. The next two columns contain a percentage breakdown of the calls of this TAC class into the following categories:
  - calls that were processed immediately (WAITTIME=0), and
  - calls that had to be placed in a TAC class-specific queue (WAITTIME>0).

- The values in the columns AVERAGE / MINIMUM / MAXIMUM WAIT TIME refer to the jobs which openUTM temporarily placed in a TAC class-specific queue. The average minimum or maximum wait time of a job per TAC class is displayed.

> The average wait time of jobs per TAC class can also be queried with the administration command KDCINF TACCLASS or with the corresponding function in WinAdmin/WebAdmin or KDCADMI while an application is running.

Wait time for dialog jobs

In the case of dialog jobs, the wait time is the period between the acceptance of the job by the application (job retrieved from the queue of the application) and the start of the program unit.

Wait times for asynchronous jobs

openUTM also records the wait time of asynchronous jobs. The wait time is defined as follows:

Asynchronous job	**Definition of “wait time”**
Input asynchronous TAC	Period between the acceptance of the job by openUTM and the start of the asynchronous service.
FPUT call in the program unit	Period between the end of the transaction in which the FPUT job was executed, and the start of the asynchronous service.
DPUT job in the program unit	Period between the conversion of the DPUT into FPUT and the start of the asynchronous service.

If the asynchronous job was not created in the current application run, the asynchronous wait time is always taken to be the time difference between the start of the application and the start of the asynchronous job.

13.2.4.7 TACCL: TAC SPECIFIC TAC CLASS EVALUATION

The TACCL list contains the same information as the TCLASS list, broken down according to the individual transaction codes. It lists all TACs that were called in the evaluation period. The TACs are listed in the sequence they first occurred. For an explanation of the individual columns, see the description of the TCLASS list format.

TAC	TAC-CLASS	NUMBER CALLS	DISTRIBUTION IN PERCENT			AVERAGE WAIT TIME (IN MSEC)
			NUMBER CALLS	WAIT TIME=0	WAIT TIME> 0	
TAC1	0	10	22.50			
TAC2	5	5	11.25	60.00	40.00	1000
TAC3	2	1	2.50	0.00	100.00	3256
TAC4	13	12	23.40	40.00	60.00	2000
TAC5	15	17	26.20	55.00	45.00	4000
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

No WAIT TIME specifications are entered for TACs of TAC class 0.

13.2.4.8 TACPT: TAC SPECIFIC DISTRIBUTION OF PROCESSING TIMES

This table lists the minimum (MIN), maximum (MAX), and mean (MEAN) processing time in milliseconds for all TACs processed within the evaluation period. It only includes the TACs whose start and end time lie within the evaluation period. The list has the following format:

TAC	PROCESSING TIME PER TAC (IN MSEC)		
	MEAN	MIN	MAX
TAC1	150000	125000	175000
TAC2	35000	19000	45000
TAC3	24500	20000	29000

The table is sorted in descending order according to the mean processing times. Only TACs with a mean processing time > 0 are displayed. The times that are output are the real times (elapsed times).

13.2.4.9 TACLIST: TAC SPECIFIC STATISTICS

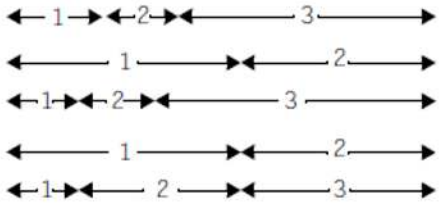
This list contains the following TAC-specific information:

- The average size of the communication area (column AVERAGE SIZE OF KB)
- The average number of database calls, broken down according to:
  - calls from the application program units (FROM USER column), and
  - calls used for coordination between openUTM and the database system (FROM SYSTEM column).
- The breakdown of processing time into:

  1: program
  2: system code
  3: data base (these times have the same meaning as in the TASKS list)

The list has the following format:

TAC	NUMBER CALLS	AVERAGE DB CALLS		AVERAGE SIZE OF KB
		FROM USER	FROM SYSTEM	
TAC1	5	15	2	256
TAC2	18	0	0	1024
TAC3	3	200	2	3000
TAC4	70	0	0	1024
TAC5	500	32	2	30000
.	.	.	.	.
.	.	.	.	.



The list is not sorted; the TACs appear in the sequence in which they first occur.

The list only includes TACs whose start and end times lie within the analysis period.

13.2.4.10 TRACE: TASK SPECIFIC TRACES

TRACE lists can be created for a more precise analysis of the execution of a UTM application. This list contains all UTM calls for the individual UTM processes in chronological order.

For space reasons, a list can contain the data from a maximum of 6 processes. If data for more than 6 processes exists for the evaluation period, the evaluation tool KDCEVAL creates a second TRACE list with the next maximum 6 processes and appends this list to the first list in the same file. It is thus possible to analyze the chronological execution of up to 12 processes.

If data from more than 12 processes exists in the evaluation period, KDCEVAL creates the evaluation lists for 12 processes. KDCEVAL does not evaluate any data for the remaining processes, and logs this fact in a UTM message specifying the TSNs.

The list is sorted in chronological order.

The TIME STAMP column contains the time stamp of the corresponding call that was logged (in microseconds).

The TRACE list records the following events and data:

- The transaction code called (TAC).
- The transaction ID. In openUTM, a unique transaction ID is assigned to each transaction. This identifier is also transferred to the attached databases on the UTM-DB interface. In this way, it is possible to link database traces with these UTM traces and establish relationships between UTM and DB processes. The transaction ID is made up of four parts:

  SC Session counter: This counts the application runs. The number is 1 after a regeneration, and is incremented by 1 each time the application starts.

  VC Service counter: This counts the services within the application run and runs up to 16 777 216 (2^{24}).

  TC Transaction counter: This counts the transactions within a service and runs up to 32 768 (2^{15}).

  VN Conversation number: this is the number of an internal UTM table for the administration of services.

  These four parts are logged after the KDCS call INIT.
  The VC and TC specifications are of interest to the user.

- All UTM calls with operation modifications. Internal UTM calls (WAIT, CONT, ...) are also listed.

  The following are also logged:

  - KCMF for KCMF-relevant calls
  - KCRN for KCRN-relevant calls
  - KCLT for PADM/DADM calls
  - In the event of an abort with PEND ER/ FR as diagnostic information:
    - the TAC of the program unit that caused the abort
    - the return codes KCRCDC and KCRRCC
    - VC and TC for the assignment to the aborted service
  - With a PEND RS as diagnostic information:
    - the TAC of the current program unit
    - VC and TC for the assignment to the service
  - With DB calls FITA and CATA:
    - VC and TC for the assignment to the service

- All database calls, if the UTM application is coordinating with a non-XA database system (generated with the KDCDEF control statement DATABASE).

  Calls to such a database system are logged with the following entry:

  opcode db-time

  Where *opcode* is the operation code of the database call according to the IUTMDB interface. See also the description of DB Diagarea in the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”, chapter UTM dump. *db-time* specifies how much time was required to process the database call (real time in microseconds).

  Example: `FITA 115`

  If SESCOS is activated for a SESAM-DBH, calls to the SESAM database are logged by the following entry, which enables the assignment between TRACE list and SESCOS evaluation:

  ppxyttttnnnmmmmmmmm

  pp

  Operation code of the database call as a hexadecimal value, e.g. 14=DBFITA.
  See
  DB Diagarea for values.

  x

  Configuration name of the SESAM-DBH.
  Value range: 'BLANK', A-Z, 0-9
  Value ‘-’: Information was not supplied by SESAM.

  y Communication name of the SESAM-DBH.
  Value range: 'BLANK', A-Z, 0-9

  tttt

  TSN of the DBH task.

  nnn

  Serial number of the SESCOS file.

  mmmmmmmm

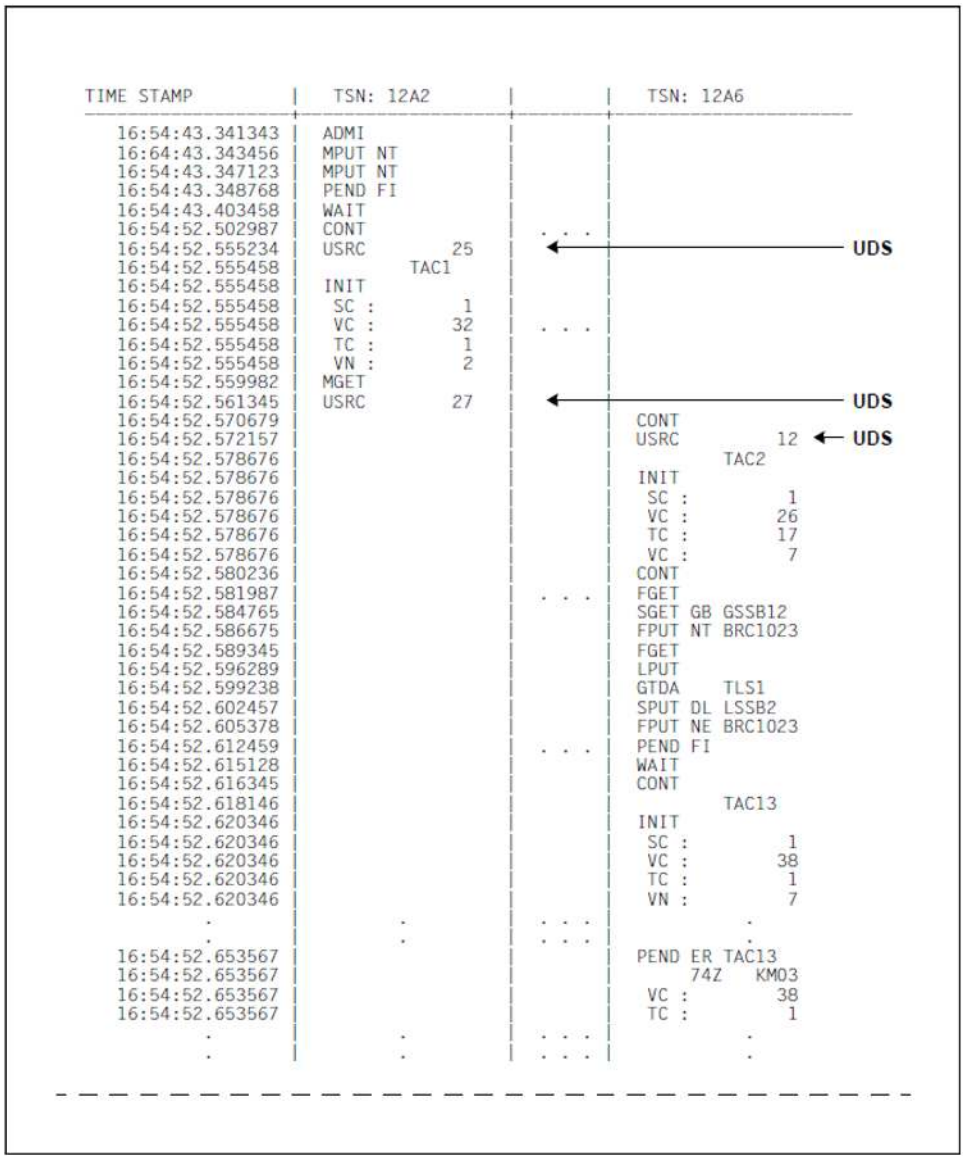
  Serial number in the SESCOS file.

  The numbers *nnn* and *mmmmmmmm* are specified with leading blanks.

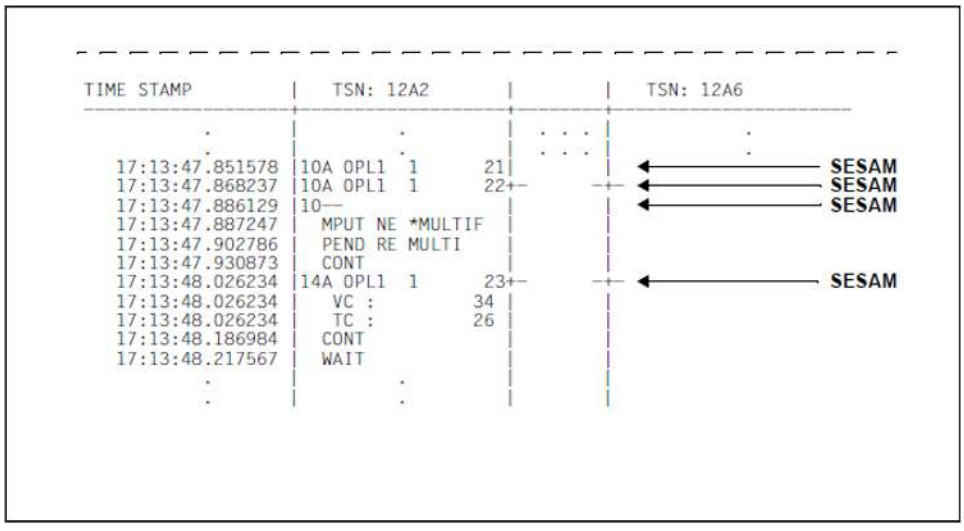
As long as no process switch takes place, all calls for processing a dialog step are listed in succession in the same TSN column. Following a PEND PA/PR/SP, a process switch can occur when changing a TAC class. The interruption of a process by the operating system can be seen by the fact that the calls are continued in another process column midway through the processing of a dialog step.

Below is an example of the TRACE list. It shows an excerpt from the execution of a UTM application with database calls to UDS and SESAM.

The TRACE evaluation list has the following format:



TRACE list (part 1)



TRACE list (part 2)

To help you understand the list, the database calls in the example are identified by arrows, which are not contained in the original TRACE list.

13.2.4.11 TRACE2: TASK PERFORMANCE TRACE

The most important events in the program units of the applications are contained in the TRACE2 evaluation list in sequence. Since the evaluation is not broken down into columns for the UTM tasks as in the TRACE list, the TRACE2 list can show any number of tasks. In addition to the entries of the TRACE evaluation, TRACE2 also contains important data for performance analysis.

The entries in the evaluation are sorted in chronological order. The TIME STAMP column contains the time stamp of the event (in microseconds).

The TRACE2 evaluation list records the following events and data:

- Start of a program unit as an entry `strt >>> tac` with
  - Transaction code of the program unit
  - TAC class
  - Current I/O and CPU stamp (in milliseconds)
  - Wait time of message BCAM
  - Wait time of job in the TAC class
- All UTM function calls with operation code and modification, plus the information:
  - KCMF for KCMF-relevant calls
  - KCRN for KCRN-relevant calls
  - KCLT for PADM and DADM calls
  - For PEND calls with KCOM = ER/FR/RS and for the database calls FITA and CATA, the transaction ID (SC, VC,TC, and VN) for the assignment to the aborted service.
  - If KCRCCC != 0, the return codes KCRCDC and KCRRCC and the transaction ID (SC,VC,TC and VN).
- All database calls, if the UTM application is coordinating with the DB system. The calls to the database system are logged with an entry DBCL *opcode db-time.*

  opcode and *db-time* mean the same as in the TRACE list.

  If SESCOS is activated at the DBH side, the information `SESCOS-INFO -->`*xytttttnnnmmmmmmmm* is also logged for calls to the SESAM database system. The values enable the assignment of TRACE2 lists and SESCOS evaluation. See also the description of the fields for the TRACE list on "TRACE: TASK SPECIFIC TRACES".
- End of the program unit as an entry `WAIT end<<<<` with
  - CPU utilization in the program unit in microseconds in the “CPU” column
  - Number I/Os in the program unit in the “I/O” column
  - At the end of column "TRACE" the first two bytes of the last bourse announcement are rendered as four characters representing the sedecimal value of the announcement.
    This information allows the system service a better analysis and diagnosis of possible performance problems.

    Example:

```
TIME STAMP    | TRACE                       | ID  TSN |
  16:39:25.002   | WAIT    end<<<<< 45C5 |  4  "       |
```

- A logical processor number supplied by KDCMON.

  Each number thus identifies a CPU. The number is indicated in the “ID” column. On multiprocessor systems, you can thereby determine which UTM tasks run in parallel.

The structure elements <<<<<< in the list make it easier to read the entries.

The TRACE2 evaluation list has the following format.

```
TIME STAMP     | TRACE                | ID TSN |   SC       VC     TC     VN|KCRN    CPU  |KCMF    I/O  |TACCLASS Q.TIME   |BCAM W.TIME
10:48:08.353714| DBCL USRC         205| 1 8E8R |                           |            |            |                  |
10:48:08.353761| CONT 31 -------------| 1 8E85 |    1   192108      1     31|            |            |                  |
10:48:08.353900| WAIT    -------------| 1  "   |                           |            |            |                  |
10:48:08.353905+ WAIT end   <<<<< 45C5+ 1  "   |<<<<<<<<<<<<<<<<<<<<<<<<<<<|        +621 |          +2 |<<<<<<<<<<<<<<   |<<<<<<<<<<<
10:48:08.353925| DBCL USRC        2065| 1 8E8R |                           |            |            |                  |
10:48:08.356002| DBCL USRC         205| 1  "   |                           |            |            |                  |
10:48:08.356213| DBCL USRC           3| 1  "   |                           |            |            |                  |
10:48:08.356219+ DBCL USRC        3370+ 1  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.357268| strt >>>RDUDS        | 1 8E85 |    1   192110      1     32|       40758 |       98786 | 0 *            0 |          0
10:48:08.357321| INIT    -------------| 1  "   |                           |            |            |                  |
10:48:08.357335| MGET    -------------| 1  "   |                           |            |            |                  |
10:48:08.357355+ DBCL USRC           8+ 1  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.357370| DBCL USRC         307| 1  "   |                           |            |            |                  |
10:48:08.357685| DBCL USRC           2| 1  "   |                           |            |            |                  |
10:48:08.357690| DBCL USRC         121| 1  "   |                           |            |            |                  |
10:48:08.357818+ DBCL USRC           1+ 1  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.357822| DBCL USRC           2| 1  "   |                           |            |            |                  |
10:48:08.357835| MPUT NT -------------| 1  "   |                           |            |            |                  |
10:48:08.357850| MPUT NT -------------| 1  "   |                           |            |            |                  |
10:48:08.357859+ MPUT NT -------------+ 1  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.357868| PEND FI -------------| 1  "   |                           |            |            |                  |
10:48:08.358905| strt        >>>WRUDS | 1 8E83 |    1   192111      1     33|       41722 |      100624 | 0 *            0 |          0
10:48:08.358975| INIT    -------------| 1  "   |                           |            |            |                  |
10:48:08.358986+ MGET    -------------+ 1  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.359007| DBCL USRC           7| 1  "   |                           |            |            |                  |
10:48:08.359018| DBCL USRC         707| 1  "   |                           |            |            |                  |
10:48:08.359214| strt        >>>WRUDS | 2 8E86 |    1   192112      1     30|       40893 |       98246 | 0 *            0 |          0
10:48:08.359274+ INIT    -------------+ 2  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.359286| MGET    -------------| 2  "   |                           |            |            |                  |
10:48:08.359304| DBCL USRC           6| 2  "   |                           |            |            |                  |
10:48:08.359317| DBCL USRC         173| 2  "   |                           |            |            |                  |
10:48:08.359496+ DBCL USRC           1+ 2  "   +---------------------------+------------+------------+------------------+-----------
10:48:08.359500| DBCL USRC          41| 2  "   |                           |            |            |                  |
10:48:08.359544| DBCL USRC        1188| 2  "   |                           |            |            |                  |
10:48:08.359614| RSET    -------------| 1 8E8R |    1   192109      1     34|            |            |                  |
10:48:08.359653+ DBCL CATA         192+ 1  "   +    1   192109      1     34+ ----------- +------------+------------------+-----------
10:48:08.359734| DBCL USRC           1| 1 8E83 |                           |            |            |                  |
10:48:08.359738| DBCL USRC         129| 1  "   |                           |            |            |                  |
10:48:08.359850| DBCL EDVG          87| 1 8E8R |                           |            |            |                  |
```

```
10:48:08.359872+ DBCL USRC       2389+ 1 8E83 +---------------------------+------------+------------+-----------------+-----------
10:48:08.359945| CONT 4C -------------| 1 8E8R |    1   192109      1     34|            |            |                 |
10:48:08.359992| DBCL USRC           5| 1  "   |                          |            |            |                 |
10:48:08.360002| DBCL USRC         251| 1  "   |                          |            |            |                 |
10:48:08.360262+ DBCL USRC         111+ 1  "   +---------------------------+------------+------------+-----------------+-----------
10:48:08.360378| DBCL USRC          34| 1  "   |                          |            |            |                 |
10:48:08.360417| DBCL USRC           2| 1  "   |                          |            |            |                 |
10:48:08.360423| DBCL USRC        4689| 1  "   |                          |            |            |                 |
10:48:08.360740+ DBCL USRC           2+ 2 8E86 +---------------------------+------------+------------+-----------------+-----------
10:48:08.360745| DBCL USRC           2| 2  "   |                          |            |            |                 |
10:48:08.360757| MPUT NT -------------| 2  "   |                          |            |            |                 |
10:48:08.360777| MPUT NT -------------| 2  "   |                          |            |            |                 |
10:48:08.360785+ MPUT NT -------------+ 2  "   +---------------------------+------------+------------+-----------------+-----------
10:48:08.360795| PEND FI -------------| 2  "   |                          |            |            |                 |
10:48:08.361494| DBCL FITA         574| 1 8E85 |    1   192110      1     32|            |            |                 |
10:48:08.362080| CONT 31 -------------| 1  "   |                          |            |            |                 |
```

14 Load simulation with Workload Capture and Replay

Thanks to the Workload Capture & Replay function, it is possible to record UTM application communications with UPIC clients and then replay these in combination with adjustable load profiles. In this way, it is possible to test the behavior of the UTM application at high loads under real-life conditions.

Workload Capture & Replay consists of the following components:

- *UPIC Capture*: Records communication with the UPIC client.

  The trace function BTRACE (BCAM trace), which is present on all the server platforms, is used to record UPIC sessions.

  It may then also be necessary to merge the traces.
- *UPIC Analyzer*: Used to analyze the recorded communication.

  Analysis is performed using the program *UPICAnalyzer* which is supplied with UPIC on 64-bit Linux systems.
- *UPIC Replay*: Used to replay the recorded UPIC session with different load parameters (speed, number of clients).

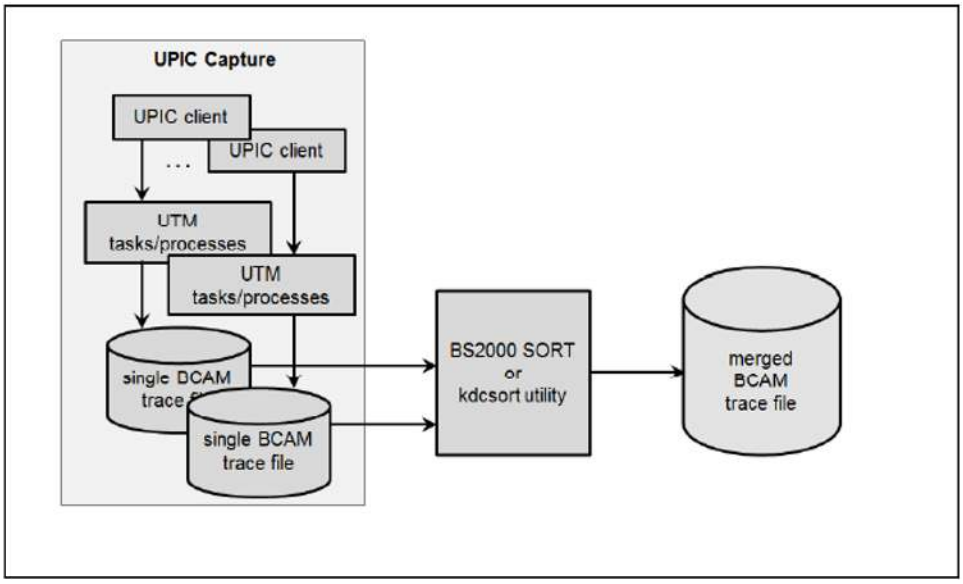
  This is done using the program *UPICReplay* which is supplied with UPIC on 64-bit Linux systems.

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

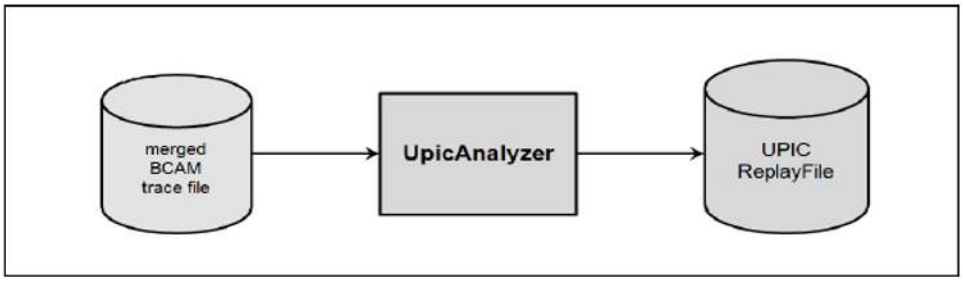
You perform the following steps to run the Workload Capture & Replay function:

1. Enable the BCAM trace and start UPIC communication, see section “Recording the UPIC conversation (UPIC Capture)”.
2. Stop the BCAM trace and merge the BCAM trace entries in a trace file (if necessary), see section “Merging trace entries”.

   These two steps are illustrated in the figure below.

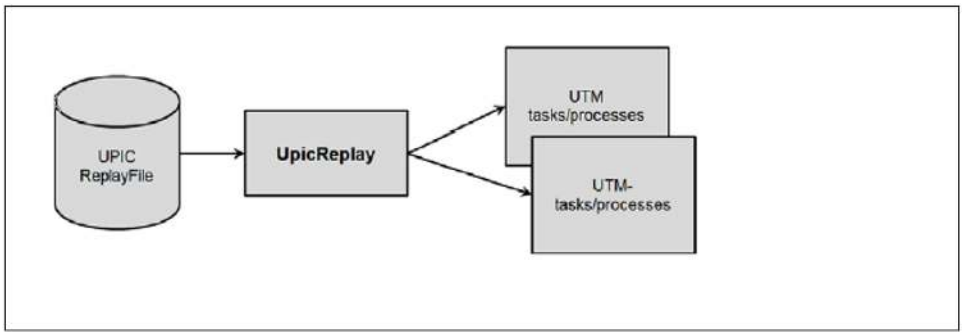


3. Perform a binary transfer of the trace file to the 64-bit Linux system where you have installed UPIC. Please note that you have to transfer the file by means of openFT.
4. Create a UPIC ReplayFile on the 64-bit Linux system on which the UPIC client is installed. To do this, call the program *UpicAnalyzer* with the trace file as input file, see the figure. For details, see section “Preparing data using the program UpicAnalyzer”.



5. Start the program

   UpicReplay with the UPIC ReplayFile as the input file, see the figure. For details, see section “Replaying the UPIC session using the program UpicReplay”.



14.1 Recording the UPIC conversation (UPIC Capture)

For this step, the UTM application can run on any UTM platform. (BS2000, Unix, Linux, or Windows system).

The UPIC clients can run on any UPIC platform. Even UPIC clients based on the product openUTM-JConnect are supported.

During this phase, the communication between the UTM application and the UPIC clients must be recorded in full and the trace length must be greater than the maximum message length. This is achieved using the UTM function BCAM trace.

> ¡ On BS2000, both the UPIC client and the UTM application support encryption when installed on the B2000 system. In this case, you cannot work without encryption.
>
> In addition, *UpicReplay* ignores the UPIC client call for setting a new password.

Please note that it must also be possible to repeat the required UTM services as often as necessary.

To do this, proceed as follows:

1. Start the BCAM trace by setting the start parameter BTRACE=ON,*length*, see "Start parameters of the application". You are recommended to specify the maximum value for *length* to prevent messages from being truncated. You can also enable the BCAM trace by means of the administration functions (KDCDIAG command or via WinAdmin/WebAdmin). In this case, however, the default value (256 bytes) is assumed for *length*, if you have not specified any other length with the BTRACE start parameter.
2. Perform the UPIC conversations between the UPIC client and the UTM application that are required for the load simulation. This also includes all aspects of establishing the connection to the UPIC clients. The associated UTM services must be fully completed at least once.
3. End the BCAM trace by means of the KDCDIAG command or via WinAdmin/WebAdmin.

This step results in binary TRACE files for all UTM processes. For details on the BTRACE files, see openUTM manual ”Messages, Debugging and Diagnostics”.

14.2 Merging trace entries

This step is necessary if the UTM application was running with more than one process during recording, a scenario that generally applies in the case of UTM applications running at medium or high load.

In this step, the binary BTRACE files of all UTM processes are sorted and entered in a common BTRACE file on the basis of their timestamps. This process step must always run on the same platform as step 1 (UPIC Capture).

On BS2000 systems, you do this using the BS2000 utility program SORT.

This step results in a sorted binary BTRACE file that contains all the trace entries in the correct temporal sequence.

You can use the provided BTRACE example procedure for sorting, see section “Sample procedures”.

14.3 Preparing data using the program UpicAnalyzer

The program *UpicAnalyzer* is supplied with UPIC on Linux (64-bit). *UpicAnalyzer* reads the trace records from a BTRACE trace, filters out the UPIC trace records, prepares these and writes them to a file in a specific format (UPIC ReplayFile Layout). This file can then be used as the input file for the program *UpicReplay*.

UpicAnalyzer is called as follows from the Linux shell:

`UpicAnalyzer` *inputfile outputfile*

Meaning of the parameters

inputfile	Name of the BTRACE file that you have transferred to the Linux system.
outputfile	Name of the output file (UPIC ReplayFile). You can use this file to play back the UPIC session with *UpicReplay*.

The program *UpicAnalyzer* recognizes the type of platform on which the trace file was created and processes the contents in the light of the platform's specific characteristics.

Example

The transferred trace file has the name *btrc.sorted*. It has to be prepared and the output written to the file *Replayfile*. The call is as follows:

```
UpicAnalyzer btrc.sorted Replayfile
```

Output:

Program "UpicAnalyzer": Version 7.0 build yyyy-mm-dd on Linux Intel ,64 Bit ,Little-Endian started

inputfile "btrc.sorted"

outputfile "Replayfile"

109 UTM BCAM trace records with 17218 bytes read.

25 UPIC replay records with 2046 bytes written.

Program "UpicAnalyzer" finished.

14.4 Replaying the UPIC session using the program UpicReplay

The program *UpicReplay* is a UPIC client program that is supplied with UPIC on Linux (64-bit). Before replaying the session, you may need to adapt the UPIC configuration and/or the generation of the UTM application.

To replay the session, you should use the same UTM platform as for recording. Exceptions are possible, see “Different platforms for Capture and Replay”.

14.4.1 Adapting the UPIC configuration and UTM generation

To perform the operation on a Linux system, you need the side information file *upicfile* containing at least one entry with the name UPREPLAY. The entry must have the prefix SD or ND. For exceptions, see “Different platforms for Capture and Replay”.

This entry must be a valid entry with the TAC of a service of the UTM application. (e.g. "DEMO"). The program *UpicReplay* uses this entry to address the UTM application. The program *UpicReplay* may set the TAC appropriately using data from the replay file.

Example of a upicfile entry

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

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

UTMTEST1 must have been generated either in MAX APPLINAME or in a BCAMAPPL statement.

Notes on UTM generation

During the UPIC Replay step, and in particular in the case of high load, the UTM application may need to permit more UPIC connections from the program *UpicReplay* than were originally present during recording. Consequently, it is advisable to use an adequately dimensioned UPIC terminal pool with multiconnect functionality for UPIC access, e.g.:

```
TPOOL LTERM=REPL,PTYPE=UPIC-R,CONNECT=MULTI,NUMBER=1000
```

In this case, up to 1000 UPIC clients can sign on simultaneously via the terminal pool.

If the UPIC Replay step runs at high load then it may be necessary to increase load-dependent generation parameters. In particular, you must pay attention to the following:

- The UTM cache must be sufficiently large (MAX CACHESIZE)
- The page pool must be sufficiently large (MAX PGPOOL)
- The number of UTM tasks must be sufficient (MAX TASKS)
- The number of permitted concurrent users must be sufficiently large (MAX CONN-USERS)

Different platforms for Capture and Replay

During replay, the data is transferred 1:1 to the UTM application. If the data includes, for example, hardware-dependent binary data, then this leads to errors if there is a change of platform. Consequently, the following applies:

- It is not possible to record a UTM application session on BS2000 and then replay this with a UTM application on a Unix, Linux or Windows system. Reason: The data in the trace file is present in EBCDIC format and conversion to ASCII is not supported in UPIC.
- It is not possible to switch between 32-bit and 64-bit platforms even within one and the same family of platforms.
- It is possible to record a UTM application session on a Unix, Linux or Windows system and then subsequently play this back using a UTM application on a BS2000 system. One prerequisite is that only pure ASCII text data is transferred during the session.

  In this case, you must enter HD as the prefix in the *upicfile* in order to ensure that the data is converted correctly between ASCII and EBCDIC.

14.4.2 Calling UpicReplay

UpicReplay plays the recorded UPIC conversations back again, see “Functioning of UpicReplay”. During this step, log messages and warnings are output to *stdout* and debugging or error messages are output to *stderr*.

UpicReplay is called as follows from a Linux shell:

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

Meaning of the parameters

InputFileName

Name of the UPIC ReplayFile that you have created with UpicAnalyzer.
Mandatory parameter.

-c<numberOfClients>

numberOfClients specifies the number of UPIC clients for which the recorded conversations are to be replayed.
Default: 1, (corresponds to -c1) i.e. only one client is simulated.

The number of captured clients is an additional limit at replay time.
The actual limit depends on the relevant system limit.

-s<speedPercentage>

speedPercentage specifies the replay speed as a percentage of the original speed. This makes it possible to simulate long and short thinking times.
Default: 100 (corresponds to *-s100*) i.e. original speed.
-s200 means 200%, i.e. twice the speed, achieved by halving the thinking time.

-d Enable debug output to *stderr*, i.e. debug messages are output on thread generation and there are fewer messages on send and receive calls.

-dd

Enables extended debug output to *stderr*, i.e. detailed debug messages are output. This option is only intended for internal *UpicReplay* diagnoses.
-dd is only of value when simulating a small number of clients.
Default: no debug output.

Example

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

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

14.4.3 Functioning of UpicReplay

Whenever possible, *UpicReplay* replays the communication exactly as it was during recording:

- A UPIC thread that replays the relevant UPIC conversation of the UPIC client is generated for each UPIC PTERM /LTERM for which a trace record is found in the UPIC ReplayFile.
- This UPIC thread runs in a loop that sends all the input messages to the UTM service in the same way as during recording, i.e. with the same data content and control flow. The procedure is similar for the retrieval of output messages from the UTM application. In this case, the content of the output messages is not checked.

Problems on replay

In the following cases, discrepancies occur between the recording and its reproduction on replay:

- Incomplete recording

  A UPIC conversation (i.e. a UTM service) was started before recording via BCAM trace was activated.

  A corresponding message is output and all input messages from this started conversation for this client are discarded.

  The UPIC client then searches the recording for the start of a new conversation for this UPIC client:

  - If a new conversation is found in the records recorded for this PTERM/LTERM then this client first waits in accordance with the recorded timestamps and then starts the load simulation from this point.
  - If no new conversation is found then this UPIC replay thread is terminated without communication with the UTM application.
- Truncated service

  During replay, a UTM service is terminated (normally or abnormally) after fewer communication steps than in the recording of the UTM application. This can occur:

  - if the application program is not able to process the recorded input data correctly because, for example, the input message contains time specifications which are rejected by the program as "late". The UTM service is therefore terminated prematurely.
  - if an impermissible UTM transfer admission is used during replay, e.g. missing UTM administration authorization.

    In this case, a corresponding message is output and the UPIC Replay thread rejects further messages until it finds a new start of conversation for this client in the recording. The UPIC thread then continues at this start of conversation following a corresponding pause time or it terminates if no further conversation is found for this UPIC client.
- Conversation too long

  On replay, a UTM service has more communication steps than during recording.

  The UPIC thread terminates this service abnormally by disconnecting the connection due to unrecorded input data. It also generates a specific warning.

  The start of the next conversation for this client is then searched for in the recording. The UPIC thread then continues at this start of conversation following a corresponding pause time or it terminates if no further conversation is found for this UPIC client.

- Incomplete input message

  An input message could not be fully recorded due to the trace record length restriction, despite an effort to compress it during recording.

  The record is rejected with a warning and the start of the next conversation for this client is searched for in the recording.

  The UPIC thread then continues at this start of conversation following a corresponding pause time or it terminates if no further conversation is found for this UPIC client.

- Other errors

  Another, unexpected return code, not covered by the cases listed above, is reported at the UPIC program interface.

  This situation can occur, for example, if the UTM application is either inaccessible or rejects the establishment of a connection.

  In such cases, the UPIC thread outputs an error message.

  The relevant UPIC thread is terminated without searching for new conversations for this client in the recording. All other UPIC conversations that are not directly affected by this problem continue to run unchanged.

15 Appendix

- XS-support of UTM applications

15.1 Installing openUTM

openUTM is normally installed using IMON. openUTM is loaded as a subsystem of BS2000 operating system. This means that various versions of openUTM can be loaded as independent UTM subsystems.

If you want to monitor the performance of your applications using SM2 or KDCMON, you must load the UTM-SM2 subsystem or the KDCMON subsystem in addition to the UTM subsystem.

Below is a summary of the specific information on openUTM, UTM-SM2 and KDCMON required for the installation.

Version dependencies to other software products such as database systems, FHS etc. and specifications on compiler versions and compatible runtime systems can be found in the Release Notice. This is supplied as an edited file on the product tape. The compiler version and runtime system information is also listed in section “Compiler versions, runtime systems, KDCDEF options”.

15.1.1 UTM system code

Components of the UTM system code

The UTM system code contains the following components:

- the base system
- the mapping module for the respective BS2000 version
- the modules for UTM-D functions
- the interface modules for the encryption function

These components are treated as separate units when loading and are loaded in the order given.

Mapping modules - execution of openUTM under various BS2000 versions

The modules of the UTM base system are independent of the operating system version. When openUTM requires the functions of the operating system, it calls them via “neutral interfaces”. The neutral interfaces are mapped to system-specific interfaces in a mapping module. This technique allows you to implement a openUTM version in various operating system versions.

A mapping module (= prelinked module) exists for each BS2000 version in which openUTM can run:

Mapping module	BS2000 version	Platform
KCYV190	BS2000 OSD/BC V10.0	/390 and x/86
KCYV200	BS2000 OSD/BC V11.0	/390 and x/86

The mapping module belonging to the BS2000 version is loaded dynamically from the library containing the UTM base system. The link to the base system is established.

Name spaces for modules of the UTM system code

Three separate name spaces exist for the object modules of the components of the UTM system code. The names for ENTRYs and EXTRNs begin with:

- KCS in the base system
- KCY in the mapping modules
- KCD in the UTM-D modules
- KCO in XAP-TP modules
- KCE in the encryption modules

Shipment and selectable units

All components for the use of the UTM-D functions are supplied as part of the openUTM V7.0 selectable unit. A license is required, however, to use the UTM-D functions.

UTM system code

The entire system code of openUTM V7.0 is supplied in the following library:

- SYSLNK.UTM.070.TPR (on servers with /390 architecture) or
- SKMLNK.UTM.070.TPR (on servers with x86 architecture)

This library contains the system code of the openUTM V7.0A base system, the mapping modules for the BS2000 versions, the system code of the UTM-D modules and the interface to encryption functionality.

The corrections for all modules of the UTM system code are contained in the REP file SYSREP.UTM.070. The REP file is created from the supplied file SYSRMS.UTM.070 during the installation with IMON. The corrections are inserted when loading the UTM subsystem.

The NOREF file SYSNRF.UTM.070 is also supplied. This file contains the names of the modules required for REP processing.

15.1.2 Installing product files

The name *DEFAULT-USERID is predefined as the installation name in the supplied SSD object for each of the UTM product files listed below.

```
SYSLNK.UTM.070.TPR
SKMLNK.UTM.070.TPR
SYSREP.UTM.070
```

The BS2000 system administrator can also change these predefined names during the installation of the UTM product files. He/she must note the following when doing so:

- The installation name of the REP file must thus begin with "SYSREP." because IMON creates the REP file from the RMS file during the installation.
- The installation name of the NRF file must be the same as the name of the REP file except that the character string ”SYSREP” is replaced by “SYSNRF".

15.1.3 Loading UTM system code

Creating entries in the subsystem catalog for openUTM

The UTM system code runs as a UTM subsystem of the BS2000 operating system. The entire UTM system code including the mapping module and the UTM-D modules is loaded by BS2000 subsystem management (DSSM).

The system administrator can modify the subsystem catalog during operation. SSD objects (subsystem declaration objects) are supplied with openUTM. Using these SSD objects, the system administrator can create an entry for the UTM subsystem in the subsystem catalog.

The following SSD objects are supplied with openUTM V7.0:

File	Implementation
SYSSSC.UTM.070.190	openUTM on BS2000 OSD/BC V10.0 (on servers with /390 and x86 architecture)
SYSSSC.UTM.070.200	openUTM on BS2000 OSD/BC V11.0 (on servers with /390 and x86 architecture)

The UTM-D modules are contained in the delivery package.

The following default settings apply in the subsystem catalog entry created by the system administrator with these SSD objects:

- Load time:
  The UTM subsystem must be explicitly loaded with the START-SUBSYSTEM command:
  `/START-SUBSYSTEM SUBSYSTEM-NAME=UTM, VERSION='07.0'`
- Installation user ID: *DEFAULT-USERID
- Load library:
  SYSLNK.UTM.070.TPR on servers with /390 architecture

  SKMLNK.UTM.070.TPR on servers with x86 architecture
- REP file: SYSREP.UTM.070

The system administrator can modify these default values using the command MODIFY-SUBSYSTEM-PARAMETER. For details, refer to the "IMON" user manual.

15.1.4 Unloading UTM system code

The entire UTM system code, including the mapping module and openUTM-D functions, can only be unloaded if no UTM application is running. To unload the openUTM system code, the system administrator must enter the following command:

```
/STOP-SUBSYSTEM SUBSYSTEM-NAME=UTM, VERSION='07.0'
```

After it has been unloaded, the UTM system code can be loaded dynamically, i.e. the UTM system code can be completely exchanged while the BS2000 system is running. This function can be used to change over to a UTM corrections version or to a new openUTM version.

The dynamic loading can be initiated using the command START-SUBSYSTEM. If it was specified in the subsystem catalog entry for openUTM that the UTM system code is to be loaded with the first start of a UTM application, the next start of a UTM application initiates the dynamic loading of the UTM system code.

15.1.5 Message files

The message file SYSMES.UTM.070 is supplied with openUTM for outputting UTM messages via the BS2000 message processing facility. When installing with IMON, the corresponding file is automatically added to the BS2000-message file.

15.1.6 REP files and RMS files

Object corrections are supplied in the form of an RMS file. From this file, the system administrator creates a REP file when installing directly with RMS. The following RMS files are supplied for openUTM V7.0, UTM-SM2, and KDCMON:

Subsystem	RMS file	REP file
UTM	SYSRMS.UTM.070	SYSREP.UTM.070
UTM-SM2	SYSRMS.UTM-SM2.*nnn*	SYSREP.UTM-SM2.*nnn*
KDCMON	SYSRMS.KDCMON.*nnn*	SYSREP.KDCMON.*nnn*

Here, *nnn* depends on the BS2000 version, as follows:

nnn=190 for BS2000 OSD/BC V10.0
nnn=200 for BS2000 OSD/BC V11.0

15.1.7 Operating several openUTM versions in parallel

Several openUTM versions can be loaded in parallel in the same BS2000 operating system and can be implemented simultaneously in production operation.

This function is particularly advantageous when changing over to a new openUTM version. You can then try out individual UTM applications with the follow-up openUTM version during production operation. The changeover of several UTM applications from one openUTM version to another within a computer can thus be tested and implemented in stages.

When operating several openUTM versions in parallel, please note the following:

- All openUTM versions must be entered in the BS2000 subsystem catalog. The entries must permit the parallel operation of several openUTM versions. The SSD objects supplied with openUTM include this permission.
- When loading the UTM system code with the command `START-SUBSYSTEM` you must specify the parameter `VERSION-PARALLELISM=*COEXISTENCE-MODE`.
- All parts of the application must be created with the files/libraries of the same openUTM version.
- UTM applications in various openUTM versions must not have the same name. openUTM prevents the start of the second application with the same name.

15.1.8 UTM-SM2 subsystem

The SM2 event monitor can store performance values for active UTM applications in the SM2 measurement file and output them to the screen. A prerequisite is that the UTM-SM2 component is installed. UTM-SM2 is included in BS2000-GA (basic configuration). UTM-SM2 is loaded as an independent subsystem of BS2000.

Components

UTM-SM2 contains the following libraries and files:

Name	Contents
SYSLNK.UTM-SM2.*nnn* SKMLNK.UTM-SM2.*nnn*	Load library for servers with /390 architecture Load library for servers with x86 architecture
SYSSSC.UTM-SM2.*nnn*	SSD object for BS2000 operating system
SYSREP.UTM-SM2.*nnn*	REP file

The identifier *nnn* stands for the UTM-SM2 version:

nnn=190 for UTM-SM2 V19.0 on BS2000 OSD/BC V10.0
nnn=200 for UTM-SM2 V20.0 on BS2000 OSD/BC V11.0

When installation is performed using IMON, the REP file is generated from the supplied fileSYSRMS.UTM-SM2.*nnn*.

Entries in the subsystem catalog

With the SSD object, the system administrator can create an entry for the UTM-SM2 subsystem in the subsystem catalog of the BS2000 during operation.

With the standard installation, IMON automatically creates the entry in the subsystem catalog for UTM-SM2.

The following default values apply in the SSD objects SYSSSC.UTM-SM2.*nnn* and SPMSSC.UTM-SM2.*nnn*:

- Load time:
  UTM-SM2 must be explicitly loaded with the command `START-SUBSYSTEM`.
- Installation user ID:
  *DEFAULT-USERID
- Load library:
  SYSLNK.UTM-SM2.*nnn* on servers with /390 architecture
  SKMLNK.UTM-SM2.*nnn* on servers with x86 architecture
- REP file: SYSREP.UTM-SM2.*nnn*

The UTM-SM2 subsystem default settings can be changed by the system administrator. For changes to be permanent, they should be made using IMON resources. For details, refer to the "IMON" user manual.

> The default value for the load time must **not** be changed by the system administrator.

Loading the UTM-SM2 subsystem

UTM-SM2 can be loaded via a system administrator command as well as via openUTM.

- Loading via a command:

  ```
  /START-SUBSYSTEM SUBSYSTEM-NAME=UTM-SM2
  ```

- Loading via openUTM (there are two possibilities here):
  - UTM-SM2 is automatically loaded at the start of the application if the application is generated with MAX SM2=ON.
  - UTM-SM2 is loaded implicitly by the administration when the data is supplied if the application is generated with MAX SM2=OFF or MAX SM2=ON.

Unloading UTM-SM2

UTM-SM2 is unloaded with the command:

```
/STOP-SUBSYSTEM SUBSYSTEM-NAME=UTM-SM2
```

15.1.9 KDCMON subsystem

The UTM event monitor KDCMON is implemented as an independent subsystem of the BS2000. You can use KDCMON to monitor UTM applications running on different openUTM versions KDCMON is included in BS2000-GA (basic configuration).

Delivery components

The libraries, files and programs listed below are required to implement KDCMON

Name	Contents
SYSLNK.KDCMON.*nnn* SKMLNK.KDCMON.*nnn*	Load library for servers with /390 architecture Load library for servers with x86 architecture
SYSSSC.KDCMON.*nnn*	SSD object for BS2000 operating system
SYSREP.KDCMON.*nnn*	REP file
SYSMES.KDCMON.*nnn*	Message file for BS2000 operating system
SYSPRG.KDCMON.*nnn*	Program to start KDCMON
SYSPRG.KDCMON.*nnn*.KDCPMSM	Program to convert and sort the performance data

Where *nnn* stands for the KDCMON version:

nnn=190 for KDCMON V19.0 on BS2000 OSD/BC V10.0
nnn=200 for KDCMON V20.0 on BS2000 OSD/BC V11.0

When installation is performed using IMON, the REP file is generated from the supplied file SYSRMS.KDCMON.*nnn* .

Installing the KDCMON subsystem

The KDCMON components supplied are stored under *DEFAULT-USERID during the installation.

Entries in the subsystem catalog

With the SSD object, the system administrator can create an entry for the KDCMON subsystem in the subsystem catalog of the BS2000 during operation.

With the standard installation, IMON automatically creates the entry in the subsystem catalog for KDCMON.

The following default values apply in the SSD objects SYSSSC.KDCMON.*nnn*:

- Load time: KDCMON is implicitly loaded with the first call.
- Installation user ID:
  *DEFAULT-USERID
- Load module library:
  SYSLNK.KDCMON.*nnn* on servers with /390 architecture
  SKMLNK.KDCMON.*nnn* on servers with x86 architecture
- REP file: SYSREP.KDCMON2.*nnn*

These presets for the KDCMON subsystem can be changed by the system administrator. For changes to be permanent, they should be made using IMON's tools. For details, refer to the "IMON" User Manual.

> The default value for the load time **cannot** be changed by the system administrator.

Loading the KDCMON subsystem

To load KDCMON you must start the program SYSPRG.KDCMON.*nnn* under the ID TSOS or explicitly with:

```
/START-SUBSYSTEM SUBSYSTEM-NAME=KDCMON
```

Unloading the KDCMON subsystem

The KDCMON subsystem is unloaded with the DSSM command:

```
/STOP-SUBSYSTEM SUBSYSTEM-NAME=KDCMON
```

If you want to unload KDCMON when data is still being recorded in one of the UTM applications, you must specify:

```
/STOP-SUBSYSTEM SUBSYSTEM-NAME=KDCMON,FORCED=YES
```

15.2 Calling UTM tools

You can start the UTM tools by means of START-EXECUTABLE-PROGRAM or using separate SDF commands.

15.2.1 Starting UTM tools via START-EXECUTABLE-PROGRAM

The UTM tools are generated as LLMs and are shipped in the program library SYSLNK.UTM.070.UTIL. Each tool is called as follows.

```
/START-EXECUTABLE-PROGRAM FROM-FILE=                                          -
/               *LIBRARY-ELEMENT(LIBRARY=$user-id.SYSLNK.UTM.070.UTIL,        -
/               ELEMENT-OR-SYMBOL=toolname)
```

toolname is the name of the tool, e.g. KDCDEF, and $*user-id* is the installation user ID (IMON installation path).

15.2.2 Starting UTM tools via separate SDF commands

The commands for the UTM tools are stored in the SDF UTM application area.

The following commands are available:

```
START-CALLUTM
START-KDCBTRC
START-KDCCSYSL
START-KDCDEF
START-KDCDUMP
START-KDCEVAL
START-KDCMMOD
START-KDCMON
START-KDCMTXT
START-KDCPMSM
START-KDCPSYSL
START-KDCUPD
START-XATMIGEN
```

Shared parameters for all commands:

MONJV =
Specifies a job variable for monitoring the program run.
Default: *NONE

MONJV = *NONE
No monitor job variable is used.

CPU-LIMIT =
Maximum CPU time in seconds that the program may take at runtime.
Default: *JOB-REST

CPU-LIMIT = *JOB-REST
The remaining CPU time is to be used for the job.

CPU-LIMIT = <integer 1..32767 seconds>
Only the specified time is to be used

VERSION =
Product version of the tool that is to be started.

VERSION = *STD
The product version is not explicitly specified.
The product version is selected as follows:

1. The version previously defined with the /SELECT-PRODUCT-VERSION command.
2. The highest openUTM version installed with IMON.

Specify the version as follows:

`<product-version> : Format mm.n<a<so>>`

mm	Main version 1..99
n	Revision version 0..9
a	Manual release A..Z (can be omitted)
so	Correction state 00..99 (can be omitted)

> As you are guided through the dialog, keep entering ? to keep this output, see also “Example 2”.

You can display the installed openUTM versions as follows:

```
/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=UTM(VERSION=*ALL), -
                        LOGICAL-IDENTIFIER=*NONE
```

Additional parameters for the START-CALLUTM command:

TRANSPORT-SYSTEM =
Specifies the transport system used by CALLUTM.
Default: *SOCKET

TRANSPORT-SYSTEM = *SOCKET
The SOCKET transport system is used.

TRANSPORT-SYSTEM = *CMX
The CMX transport system is used.

You can display the complete syntax using the following commands:

```
/START-SDF-A
//OPEN-SYNTAX-FILE $.SYSSDF.UTM. 070, MODE = *READ,TYPE=*SYSTEM
//SHOW OBJECT = *DOM(UTM),SIZE=*MAX,IMPL-INFO=*NO(LANGUAGE=D)
//END
```

Example 1

Calling the CALLUTM tool via a separate SDF command:

```
/START-CALLUTM?
```

```
COMMAND  : START-CALLUTM
------------------------------------------------------------------------------
TRANSPORT-SYSTEM     = *SOCKET
VERSION              = *STD
MONJV                = *NONE
CPU-LIMIT            = *JOB-REST
------------------------------------------------------------------------------
NEXT = *EXECUTE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

Example 2

Calling the KDCDEF tool via a separate SDF command:

```
/START-KDCDEF?
```

If you now enter “?” twice in the VERSION field then you will see the following output

```
COMMAND  : START-KDCDEF
OPERANDS : VERSION=*STD
------------------------------------------------------------------------------
VERSION              = *STD
                       *STD or product-version
                       product version for KDCDEF mm.r<u<so>>
                       Manual release u and Correction state so are optional
                       - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                       <product-version> : format mm.r<u<so>>
                        mm : Main version 1..99
                        r : Revision version 0..9
                        u : Manual release A..Z
                        so : Correction state 00..99
MONJV                = *NONE
CPU-LIMIT            = *JOB-REST
------------------------------------------------------------------------------
NEXT = *EXECUTE
*EXECUTE"F3" / Next-cmd / *CONTINUE / *EXIT"K1" / *EXIT-ALL"F1" /     *TEST"F2"
```

15.3 Memory classes of a UTM application

For the storage space required by the UTM product files and of a UTM application, refer to the release notice.

The table below lists the components that require address space in one of the memory classes 4, 5 or 6 while UTM applications are running:

Component	Class	Explanation
openUTM V6.2 system code	4	1.
Static KAA tables	5	2.
Dynamic KAA tables	5	3.
Cache	(5)	4.
KTA tables	5	5.
ROOT system modules	6	
Additional requirement for formatting	6	6.
Additional requirement for DB system	6	7.
Administration program KDCADM	6	
Program units of the user	6	8.

Explanation

1. openUTM system code:
   The UTM system code, including the operating system adaptation module, is loaded by BS2000 subsystem management DSSM in the class 4 memory. The UTM base components, including the operating system-adapter module and the UTM-D code including XAP-TP, belong to the UTM system code. Further details can be found in section“Installing openUTM”.
2. Static KAA tables (**K**DC **A**pplication **A**rea):
   KDCDEF outputs message `K450 KDCFILE generated; KAA-size:nnnK;...` in the UTM generation to indicate that the static KAA tables occupy *nnn* KB. When the application starts, openUTM reads these tables into a memory pool in class 5 memory. The memory pool occupies *nnn* KB, rounded off to a multiple of 1 MB.

   The inclusion of a user and a terminal results in an additional requirement of (at least) 1 KB. This value increases for “extras” like stacking, TLS, ULS, etc. To allow for the space requirement by extending the configuration, it is better to enter the number of objects with RESERVE statements in the KDCDEF input and let KDCDEF calculate the space requirement itself.
3. Dynamic KAA tables:
   The space requirement for a terminal is at least 1 KB. openUTM first uses the section at the end of the memory pool, which was created for the static KAA tables. If this space is not sufficient, openUTM creates one or more additional memory pools for the dynamic tables in class 5 memory.

4. Cache:
   All processes of a UTM application use the cache as a buffer area for I/Os to the page pool on the KDCFILE. openUTM creates the cache as a memory pool in class 5 memory of the program space, or in one or more data spaces. By creating the cache in a data space, the program space can be released. Location and size of the cache are determined by the KDCDEF parameters MAX CACHESIZE=(*number*,..., PS | DS) and BLKSIZE= *blocksize*.

   Example
   If *blocksize*=2K, its size is *number* * 2KB, rounded off to a multiple of 1 MB.
5. KTA tables (**K**DC **T**ask **A**rea):
   This area contains process-specific administrative data and is created in the class 5 memory. It is at least 8 KB long. The space requirement increases in accordance with the KDCDEF generation parameters.
6. Additional requirement for formatting:
   The code of the format handling system is loaded dynamically. For information on the memory requirement for the format handling system, see the description of FHS. The memory requirement for the formats is added to this.
7. Additional requirement for database system:
   The connection module of the database system can be linked statically or dynamically. The memory requirement can be found in the information for the respective database system.
8. Program units of the user:
   The memory requirement is indicated in the linkage editor list.

15.4 Compiler versions, runtime systems, KDCDEF options

In this section you will find information on which versions of the compiler and runtime systems of the individual programming languages you can use to create UTM program units. The following data is listed in a table for each programming language supported by openUTM:

- versions of the compiler that you can use to create the objects (OM or LLM) of a UTM program unit
- versions of the runtime system that are suitable for these program units
- values for the COMP operand of the PROGRAM control statement that you must specify in the KDCDEF generation to add these program units to the application configuration

In particular, you can determine from the table which combinations of compiler version, version of the runtime system and COMP values are allowed.

You will also find information on the compiler and runtime system versions allowed in the CRTE manual and in the documentation for the various compilers (Release Notes, manuals).

Note that compiler versions are listed in the following for which support has already been set. The reason for this is that there are older programs available as objects in some customer applications, and these objects will continue to be used.

You should generally use the compiler and runtime system versions that are still supported for all further developments and new developments! Otherwise you may not place warranty claims or have the right to receive corrections when a problem arises.

In general, the notes and restrictions in the Release Notes and compiler manuals are to be observed.

> openUTM V7.0 requires a specific CRTE version. Please refer to the release notes for more details.

Mixing languages in an application

Mixing languages means that a UTM application can be compiled from program units which were generated with compilers using different languages and which thus use different runtime systems when operating. The following should be noted, however:

A UTM application is only permitted to contain **one** runtime system for **each** programming language that is supported by openUTM and must be generated with
PROGRAM ... COMP NOT EQAL ILCS.

Mixing languages in a program unit

openUTM also allows you to mix languages within a single program unit, i.e. program units can consist of several source codes written in different programming languages. A requirement for this to function is that the program components are called in accordance with the same logical operand conventions; see the CRTE manual. In particular, the data displays when transferring parameters and when accessing shared data structures must be identical.

> **CAUTION!**
>
> The instructions in section “Information for applications with ILCS program units” are to be followed closely.

The following cases arise when calling Assembler subroutines:

- The calling program as well as the subroutine called are ILCS programs. In this case, there are no restrictions to observe.

  It is possible to program Assembler ILCS programs with Z-macros. See the openUTM manual “Programming Applications with KDCS for Assembler”.
- The calling and the called program are not ILCS programs, but they observe the same logical operation conventions. The logical operation conventions are described in the user manuals for the corresponding programming languages.

Errors in the logical language operations or address errors can arise when switching to a different language in the following constructions:

- C function calls
- Assembler function calls with @-macros
- calls to third-party software
- database calls
- formatting system calls

15.4.1 Assembler

Compiler	Runtime system	PROGRAM..., COMP=
ASSEMBH V1.3	ASSEMBH V1.3	ILCS

15.4.2 C/C++

Compiler	Runtime system	PROGRAM..., COMP=
C/C++ V3.2A	CRTE[1] V10.0A on BS2000 OSD/BC V10.0 CRTE[1] V11.0A on BS2000 OSD/BC V11.0	ILCS [2]

Comments on the table

[1]We also strongly recommend that you always use the latest corrections version for compilers and runtime systems.

[2]You can also set COMP=C. However, KDCDEF overwrites this value with COMP=ILCS without warning.

15.4.3 COBOL

Compiler	Runtime system	PROGRAM ..., COMP=
COBOL85 V2.3A	CRTE[1] V10.0A on BS2000 OSD/BC V10.0 CRTE[1] V11.0A on BS2000 OSD/BC V11.0	ILCS / COB1[2]
COBOL2000 V1.5A or later	CRTE[1] V10.0A on BS2000 OSD/BC V10.0 CRTE[1] V11.0A on BS2000 OSD/BC V11.0	ILCS / COB1 [2]

Comments on the table

[1]We also strongly recommend that you always use the latest corrections version for

compilers and runtime systems.

[2]The objects generated by COBOL85 or COBOL2000 are equally suitable for ILCS

linking and non-ILCS linking; there is no compiler option for specifically creating ILCS or non-ILCS objects.

COMP=ILCS means that ILCS linking is used. COMP=COB1 means non-ILCS linking (= old-style linking).

The value of COMP= can be set on the basis of the following criteria:

- COMP=ILCS is specified if all activated subroutines also have ILCS capability.
- COMP=COB1 must always be specified if the Cobol module calls subroutines that do not have ILCS capability.

Compiler options

The table below provides an overview of which COMOPT parameters must be set in accordance with the compiler used, and what must be noted in the process.

COMOPT parameter	Compiler	Comment
CHECK-CALLING-HIERARCHY=NO	COBOL85 COBOL2000	Only if COMP=COB1, i.e. with non-ILCS linking.
MARK-LAST-PARAMETER=YES	COBOL2000	Recommended

Mixing Cobol programs

If a program unit is generated with COMP=COB1, it may comprise modules compiled with the COB1, COBOL85, or COBOL2000 compiler. If the program unit is generated with COMP=ILCS, it can only comprise modules compiled with the COBOL85 or COBOL2000 compiler.

Within an application, program units generated with COMP=ILCS can coexist with program units generated with COMP=COB1.

15.4.4 Fortran

Compiler	Runtime system	PROGRAM ...,COMP=
FOR1 V2.2C [1]	FOR1 V2.2	ILCS

Comments on the table

[1]For the FOR1 V2.2C compiler, ILCS capability is set using the compiler option:

- If the program unit was compiled with the compiler option COMOPT LINKAGE=FOR1-SPECIFIC, then COMP=FOR1 must be specified in the PROGRAM statement. This compiler option specifies that the program uses the “old” FOR1 interface.
- If the program unit was compiled with the compiler option COMOPT LINKAGE=STD (default value), then COMP=ILCS must be specified in the PROGRAM statement. compiler option specifies that the program uses the ILCS interface.

Notes on mixing languages

A UTM application may contain FOR1 program units that have been entered in the configuration with different values for PROGRAM ...,COMP= . In this manner, program units can co-exist in a UTM application where one program unit uses the “old” FOR1 interface and another uses the ILCS interface.

15.4.5 Pascal

Compiler	Runtime system	PROGRAM...,COMP=
Pascal-XT V2.2B or later	V2.2	ILCS

15.4.6 PL/I

Compiler	Runtime system	PROGRAM...,COMP=
PLI1 V4.2	[1]	PLI1
PLI1 V4.2 or later [2]	[1]	ILCS

Comments

[1]Please note that the compiler objects must be of the same version as the runtime system used. You may not link several PLI1 runtime systems in a UTM application.

[2]You can also create ILCS program units with PLI1 versions >= V4.2A. You will need to specify OPTIONS(ILCS) in the PROCEDURE statement to do this. You must then make sure that your UTM application only contains PL/I program units that are ILCS-compatible.

15.4.7 SPL4

Compiler	Runtime system	PROGRAM...,COMP=
SPL4 V2.9A or later	[1]	SPL4 / ILCS [2]

Comments on the table

[1]The SPL V2.9A runtime system is supplied with openUTM. However, this may change with a corrections update. Please read the latest Release Notice for more information.

[2]The ILCS capability of the program unit depends on the compiler option:

- If the program unit is compiled with compiler option GEN=(ILCS=NO) (non-ILCS linking), it must be generated with COMP=SPL4.
- If the program unit is compiled with compiler option GEN=(ILCS=YES) (ILCS linking), it must be generated with COMP=ILCS.

15.4.8 Notes on upgrading from an older openUTM version

Check the following points to identify which COMP=... value is appropriate for a program unit:

- Which compiler was used to compile the program unit? Which version of the compiler was used? Which compiler options were set?
  Was an object with ILCS capability thus created?

  For more clarification, please read the information on the individual compilers provided from "Assembler" onwards.
- Does the program unit call subroutines? Do these subroutines have ILCS capability?

  If both the program unit and all subroutines (if any) have ILCS capability, the program unit should be generated with COMP=ILCS.

  If the program unit or any of the subroutines do not have ILCS capability, or if this cannot be established, please proceed as follows:
  - In the case of non-Cobol program units, the previous COMP=... value should be retained initially.
  - In the case of Cobol program units, set COMP=COB1 even if the program was compiled with the COBOL85 compiler. The former Cobol RTS must be provided when linking the UTM application program.

> **i** It is thus **strongly advisable** to change over to ILCS linking when upgrading an older UTM application. The advantage of ILCS linking is that programs in various programming languages can be activated together without problems, and that the maintenance and further development of the compilers and runtime systems are secured.

15.5 Structure of the accounting records of openUTM

The accounting records of openUTM are written in the BS2000 accounting file and are evaluated with RAV. The records comply with the conventions for BS2000 accounting records. The first 24 bytes are therefore reserved for BS2000 accounting, and the remaining data contains UTM-specific information.

The following two record types exist:

- Accounting records (UTMA record type)
- Calculation records (UTMK record type)

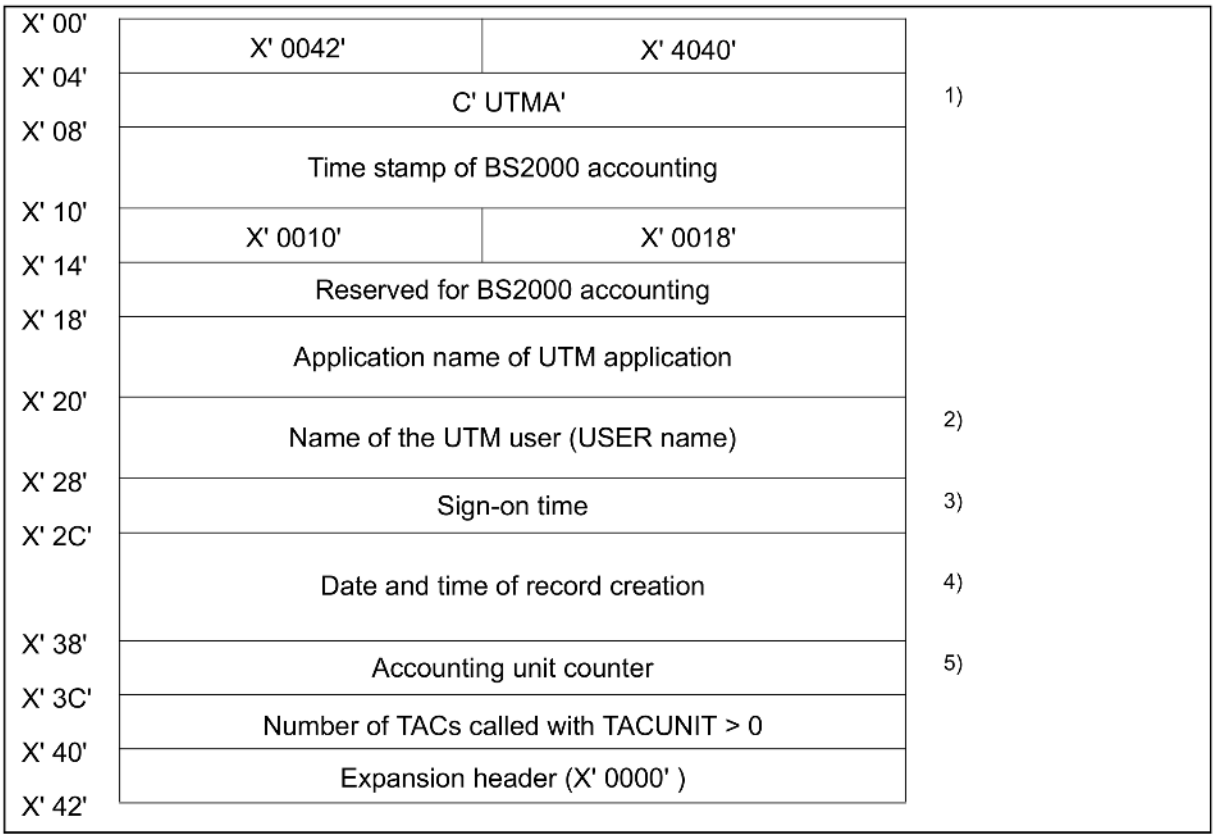
These records are described in chapter “Accounting”.

You will need the BS2000 macro ARDS to determine the structure of the accounting and calculation records in the BS2000 accounting file.

If ARDS is called with UTM=NEW, it uses the UTM macros KDCUTMA and KDCUTMK, which are located in the UTM macro library SYSLIB.UTM.070.ASS. They describe the structure of the accounting records and calculation records.

If ARDS is called with UTM=OLD, then the BS2000 macros ARDSUTMA and ARDSUTMK are used.

15.5.1 Structure of an accounting record



Explanation of comments:

[1)]Identifier of the accounting record in BS2000 accounting.

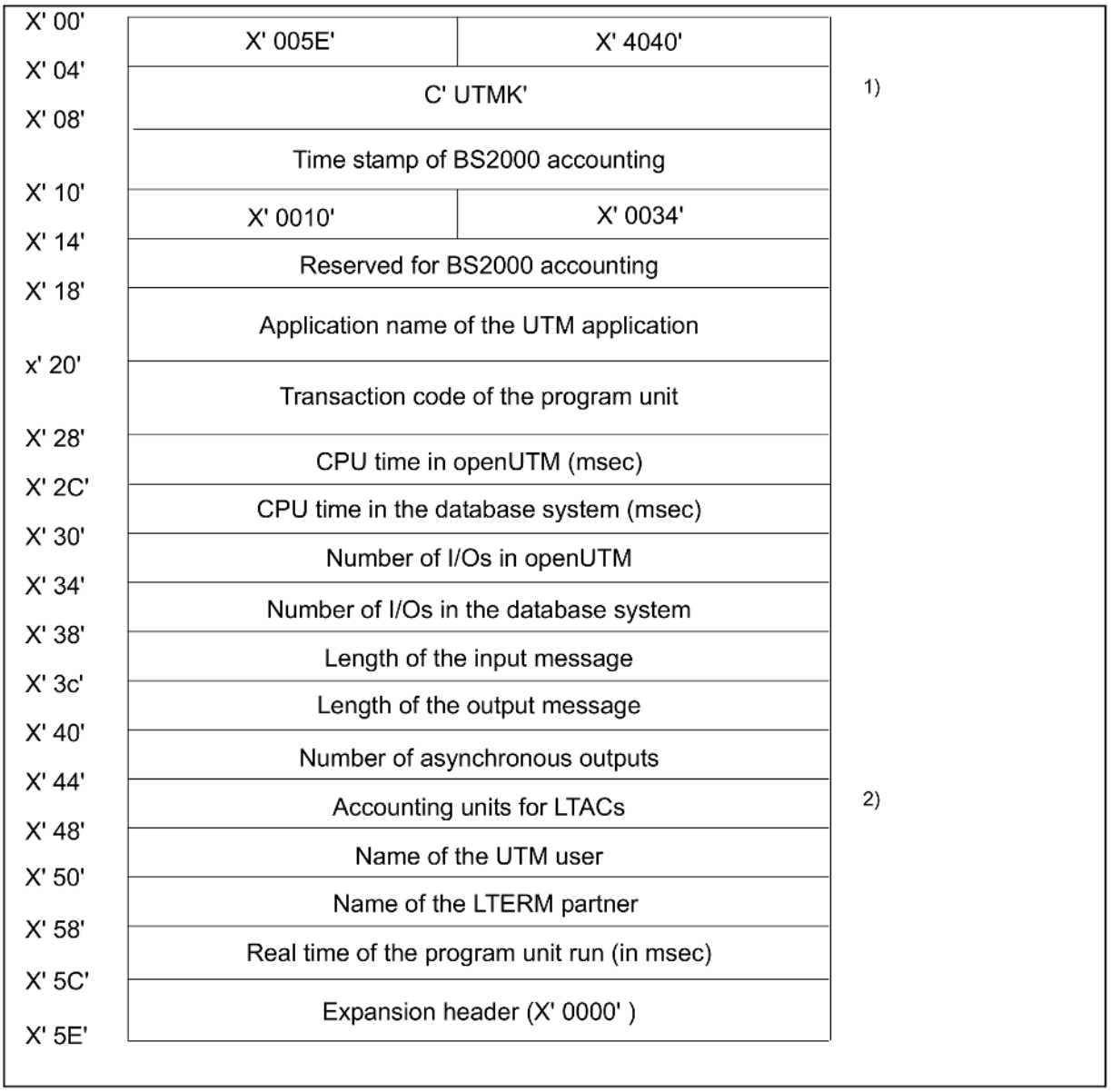
[2)]Name of the user. In a UTM application without generated users, openUTM enters the name of the LTERM partner.

[3)]Sign-on time for this user (USER) to this LTERM in seconds, relative to the time base of the BS2000 version. If only asynchronous TACs were called for this USER in the current run of the UTM application, this field contains binary zero.

[4)]Format: *yymmddhhmmss* (year/month/day/hour/minute/second)

[5)]Sum of the accounting units for this user since the last accounting record was written or since the sign-on time.

15.5.2 Structure of a calculation record



Explanation of comments:

[1)]Identifier of the record in BS2000 accounting.

[2)]See KDCDEF statement LTAC...,LTACUNIT=.

15.6 Structure of SAT log records

Security-related UTM events can be logged using the BS2000 function SAT (**S**ecurity **A**udit **T**rail). These UTM events are logged with SAT for auditing purposes. When SAT logging is switched on, minimum logging is implemented. Other events can also be defined. The logging of such events in the SAT log records can be switched on and off for specific events, specific users and specific jobs.

15.6.1 Meaning of the log data fields used by openUTM

openUTM creates a SAT log record for each event. Each log record transferred by openUTM to SAT comprises a part with a fixed structure and length, the SAT header, followed by a part with variable structure and length.

The SAT header contains the date and time, the BS2000 user, the TSN, the current BS2000 event, and its result. Only the following fields are defined by openUTM:

Field name	Meaning	Type
EVT	Type of event	C string (1..3) with openUTM, always “TRM”
RES	Result of event	Set: Success/Failure (“S” / “F”)

The variable part contains individual UTM-specific data fields, each of which is preceded by a length field and a SAT identifier. The type and number of the individual data fields depends on the type of UTM event.

The following tables indicate the log data fields used by openUTM (arranged alphabetically), the meaning of the field contents, and its data type.

UTM events can be linked with the ALARM function of SAT. Apart from a few exceptions, the data type for SAT-ALARM matches the data type for SATUT. In the following table, the ALARM data type is specified in brackets () if it differs from the data type for SATUT.

Field name	Meaning	Type
ACCTYP	Access type of UTM storage area	Set: C / D / READ / WRITE
APPLNAM	BCAM application name	C string (1..8)
CALLER	Address of caller	X string (1..4) (ALARM: X string 4..4)
COMMAND	Name of UTM-SAT administration command or administration program interface	C string (1..8)
DATNAM1	Name of UTM storage area	C string (1..8)
DATNAM2	Name of UTM object	C string (1..8)
DATTYP	Type of UTM storage area	Set: G / T / U
LTERM	LTERM partner name for clients and printers	C string (1..8)
MUXLTRM	MUX transport connection	C string (1..8)
OBJECT1	Name of UTM object or administration program interface: name of modified field	C string (1..8)
OBJECT2	Name of UTM object or administration program interface: content of modified field	C string (1..8)

OBJECT3	Administration program interface: object type	C string (1..8)
PTERM	BCAM name	C string (1..8)
TACIDEN	Transaction identifier	Set: G/T/D/C/P
TACNAM	Transaction code	C string (1..8)
USER2	UTM user name	C string (1..8)
UTMAPPL	UTM application name	C string (1..8)
UTMHEX3	RC of administration program interface	C string (1..8)
UTMNAME	Name of load module or administration program interface: name of UTM object	C string (1..32)
UTMOBJ4	Program load mode or administration program interface: subopcode1 or “FORMATTR” or field name	C string (1..8) (ALARM: C string 8..8)
UTMOBJ5	Administration program interface: format attribute or field name	C string (1..8)
UTMOBJ6	Load module version or long processor name	C string (1..24) (ALARM: C string 64..64)
UTMREAS	KDCS return code [1] Administration program interface: blank	C string (1..8)
UTMSTAT	Transaction status	C string (1)
UTMSUBC	UTM event	Set: CHANGE-PW / SIGN / DATA-ACCESS / ADM-CMD / START-PU / END-PU / TASK-ON / TASK-OFF / SEL-CMD/ CHG-PROG
UTMTAID	Transaction identification [2]	X string (1..4) (ALARM: X string 4..4)
UTMUSER	UTM user name	C string (1..8)

[1]With the UTM events CHANGE-PW, START-PU, END-PU and DATA-ACCESS, the return code is made up of the compatible and the incompatible KDCS return code.

[2]The transaction identification (TA-ID) comprises a 2-byte service counter (within a session) and a 2-byte transaction counter (within a service). It is made available at the IUTMDB interface to the database for SAT logging of database events. The transaction identification is used to assign an event to the transaction which created it.

15.6.2 Defining the data fields

This section describes the log data fields that are defined when logging the individual events.

The following tables provide an overview of which log data fields are defined in accordance with the events. The log data fields are listed in the sequence in which they appear in the SAT log record. The field contents are then explained for the individual events.

Meaning of the entries in the following tables:

—	Field not defined
Y	Field defined (mandatory fields in the context of SAT)
O	Field defined in certain cases

Field name	UTM events									
	TASK-ON	TASK-OFF	SIGN	CHANGE-PW	START-PU	END-PU	DATA-ACCESS	ADM-CMD	SEL-CMD	CHG-PROG
APPLNAM	-	-	O	-	-	-	-	O	-	-
UTMUSER	-	-	Y	Y	Y	Y	Y	Y	Y	-
UTMAPPL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
UTMSUBC	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
LTERM	-	-	Y	-	O	Y	-	O	-	-
PTERM	-	-	Y	-	-	-	-	O	-	-
MUXLTRM	-	-	O	-	-	-	-	O	-	-
DATNAM1	-	-	-	-	-	-	Y	-	-	-
DATNAM2	-	-	-	-	-	-	O	O	O	-
DATTYP	-	-	-	-	-	-	Y	-	-	-
ACCTYP	-	-	-	-	-	-	Y	-	-	-
COMMAND	-	-	-	-	-	-	-	Y	Y	-
OBJECT1	Y	Y	Y	Y	Y	Y	-	O	O	-
OBJECT2	-	-	-	-	O	-	-	O	O	-
OBJECT3	-	-	-	-	-	-	-	O	-	-
CALLER	-	-	-	-	-	-	-	-	-	-
TACNAM	-	-	-	Y	O	Y	Y	Y	Y	-
TACIDEN	-	-	-	-	Y	Y	-	-	-	-
USER2	-	-	-	-	-	-	-	O	O	-

UTMNAME	-	-	-	-	-	-	-	O	-	Y
UTMTAID	-	-	Y	Y	Y	Y	Y	Y	Y	Y
UTMSTAT	-	-	-	-	-	Y	-	-	-	-
UTMREAS	-	-	Y	Y	Y	Y	Y	Y	Y	-
UTMOBJ4	-	-	-	-	-	-	-	O	-	Y
UMTOBJ5	-	-	-	-	-	-	-	O	-	-
UTMOBJ6	-	-	Y	-	-	-	-	O	-	Y
UTMHEX3	-	-	-	-	-	-	-	Y	-	-

The structure of the variable part of the log record is described in detail below for each event.

TASK-ON: Connect a task to the UTM application

UTMAPPL	Name of the active application.
UTMSUBC	TASK-ON
OBJECT1	F for the first task or N for a follow-up task (next task) or L is initiated with a program exchange or with PEND ER (load program).

TASK-OFF: Sign off a task from the UTM application

Only the normal termination of a task is logged.

UTMAPPL	Name of the active application.
UTMSUBC	TASK-OFF
OBJECT1	Last task: yes “Y” or no “N”

SIGN: Sign on a UTM user

APPLNAM	BCAM application name.
UTMUSER	Name of the user/client that initiates the event.
UTMAPPL	Name of the active application.
UTMSUBC	SIGN
LTERM	Name of the LTERM partner via which the user/client connects to the application.
PTERM	BCAM name of the user/client assigned to defined LTERM partners.
MUXLTRM	PTERM name of the MUX transport connection.

UTMTAID	Transaction identification or zero.
OBJECT1	BCAM processor name.
UTMREAS	KCRSIGN return field.
UTMOBJ6	Long processor name.

CHANGE-PW: Change password

CHANGE-PW is also initiated if a user password is modified by the UTM administrator.

UTMUSER	Name of the UTM user who initiated the event (possibly administrator ID).
UTMAPPL	Name of the active application.
UTMSUBC	CHANGE-PW
OBJECT1	Name of the user ID whose password is changed.
UTMREAS	KDCS return codes.
TACNAM	Transaction code of active program unit.
UTMTAID	Transaction identification or zero.

START-PU: Create a job or start a program unit

UTMUSER	Name of the UTM user who initiates the event.
UTMAPPL	Name of the active application.
UTMSUBC	START-PU (start program unit).
LTERM	Name of the defined LTERM partner or blank.
OBJECT1	If TACIDEN=G: TAC of the job.
	If TACIDEN=C/T/P: TAC of the active service.
OBJECT2	If TACIDEN=G: DPUT identification of the job created.
UTMREAS	KDCS return codes (if TACIDEN=G).
UTMTAID	Transaction identification of the active transaction or zero.
TACNAM	TAC of the active program unit.
TACIDEN	Transaction identifier. Possible values (the values G, C, T, P are mutually exclusive):

	G for generated job Dialog jobs are only logged as generated if they cannot be started immediately due to TAC class control.
	C for start of conversation A conversation begins with this program unit.
	T for start of transaction A follow-up transaction of a conversation begins with this program unit.
	P for start of a follow-up program unit within a transaction

The creation of messages to an LTERM partner is not logged.
A confirmation job (to a TAC) is only logged as generated when, on the basis of the result of the main job run, it is selected and converted to a main job. The destination of the executed main job is then output in the field LTERM or TACNAM.

END-PU: Terminate a program unit

UTMUSER	Name of the UTM user who initiates the event.
UTMAPPL	Name of the active application.
UTMSUBC	END-PU (end program unit).
LTERM	Name of the defined LTERM partner or blank.
OBJECT1	Transaction code of the active service.
UTMSTAT	The field is only defined if TACIDEN=T or C. It then contains the transaction status:
	C: Transaction logging (Commit)
	R: Roll back the transaction
UTMTAID	Transaction identification.
TACNAM	Transaction code of active program unit.
TACIDEN	Possible values:
	C: End of the program unit and of the conversation.
	T: End of the program unit and of the transaction; the conversation is continued.

	P: End of the program unit; the transaction is continued.
UTMREAS	KDCS return codes.

DATA-ACCESS: Access to a UTM storage area

UTMUSER	Name of the UTM user who initiates the event.
UTMAPPL	Name of the active application.
UTMSUBC	DATA-ACCESS
DATNAM1	Name of the UTM storage area addressed.
DATNAM2	If DATTYP=ULS: UTM user
	If DATTYP=TLS: LTERM partner for clients and printers, otherwise blank
DATTYP	Type of storage area:
	G: GSSB
	U: ULS
	T: TLS

ACCTYP	Type of storage access:
	READ: Read access
	WRITE: Write access
	C: Create
	D: Delete
UTMREAS	KDCS return codes.
UTMTAID	Transaction identification.
TACNAM	Transaction code of active program unit.

ADM-CMD: Call the administration program interface

UTMUSER	Name of the UTM user who initiates the event.
UTMAPPL	Name of the active application.
UTMSUBC	ADM-CMD
COMMAND	DADM (KDCS call) or call of the program interface for administration:
	CHNGAPPL / CREATE / CREASTMT / DELETE / DUMP / ENCRYPT /

	GETOBJ / LOCKMGMT / MODIFY / ONLIMP / PETTA / SENDMSG /
	SHUTDOWN / SPOOLOUT / SYSLOG / UPDIPADR / USLOG
UTMHEX3	Return code of administration program interface.
UTMTAID	Transaction identification.
USER2	Session, user or blank.

The fields DATNAM2, LTERM and OBJECT1 may be output as blank.

> With some function calls of the program interface, various actions are possible. In this case, SAT writes a log record for each action and logs the parameters, whereby the parameter value is only output if the parameter has changed.
> See also the openUTM manual “Administering Applications”.

In certain cases, the following additional fields are defined, depending on COMMAND:

- COMMAND: CHNGAPPL
  UTMOBJ4: Subopcode1 (NEW/OLD for PROGRAM)
- COMMAND: CREATE
  OBJECT3: Object type
  UTMNAME: Object name
- COMMAND: DELETE
  OBJECT3: Object type
  UTMOBJ4: Subopcode1 (DELAY/IMMEDIAT)
  UTMNAME: Object name
- COMMAND: ENCRYPT
  OBJECT1: Subopcode1 (CREATEK, ACTIVATK, DELETEK, REAACTK, REANEWK)
- COMMAND: LOCKMGMT
  OBJECT1: Subopcode1
- COMMAND: MODIFY
  OBJECT3: Object type or parameter type
  Additional fields are logged, depending on the object type or parameter type:

Object type	**Logged fields**
CLNODE	OBJECT1: Parameter UTMOBJ6: Parameter value
CON	UTMOBJ6: Long processor name
KSET	OBJECT1: Keys

LOADMODU	OBJECT1: “Version” UTMOBJ6: Version UTMNAME: Name of load module to be modified
LPAP	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of LPAP partner to be modified
LSES	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of LSES and CON triplet to be modified UTMOBJ6: Long processor name
LTAC	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of LTAC to be modified
LTERM	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of LTERM partner to be modified UTMOBJ4: „FORMATATTR“ UTMOBJ5: Formata ttribute
MUX	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of MUX triplet to be modified
OSICON	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of OSI-CON to be modified
OSILPAP	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of OSI-LPAP partner to be modified
PTERM	PTERM: Name of client/printer to be modified (PRTM) OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of PTERM triplet to be modified UTMOBJ6: Long processor name
TAC	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of TAC to be modified UTMOBJ6: Parameter value
TACCLASS	OBJECT1: Parameter OBJECT2: Parameter value UTMNAME: Name of TACCLASS to be modified

TPOOL	LTERM: LTERM prefix OBJECT1: Parameter1 OBJECT2: Parameter value1 UTMOBJ4: Parameter2 UTMOBJ5: Parameter value2 UTMNAME: Name of TPOOL to be modified (LTERM prefix, PRONAM, PTYPE, BCAMAPPL)
USER	OBJECT1: Parameter OBJECT2: Parameter value UTMOBJ4: “FORMATTR” UTMOBJ5: Format attribute UTMNAME: Name of USER to be modified

Parameter type	**Logged fields**
CCURRPAR	OBJECT1: Parameter UTMOBJ4: Parameter value 1 UTMOBJ5: Parameter value 2
CLPAR	OBJECT1: Parameter UTMOBJ4: Parameter value
CURRPAR	OBJECT1: Parameter UTMOBJ6: Parameter value
DIAGACCP	OBJECT1: Parameter OBJECT2: Parameter value 1 UTMOBJ6: Parameter value 2
MAXPAR	OBJECT1: Parameter OBJECT2: Parameter value
TASKSPAR	OBJECT1: Parameter OBJECT2: Parameter value
TIMERPAR	OBJECT1: Parameter OBJECT2: Parameter value

- COMMAND: ONLIMP

  OBJECT1: „KC_ALL“
- COMMAND: SENDMSG

  LTERM: LTERM name or “KDCALL”
- COMMAND: SHUTDOWN

  OBJECT1: Subopcode1

  OBJECT2: Parameter value

- COMMAND: SPOOLOUT

  OBJECT1: “SPOOLOUT”

  OBJECT2: “ON”
- COMMAND: SYSLOG

  OBJECT1: Subopcode1

  UTMOBJ6: Parameter value
- COMMAND: UPDIPADR

  OBJECT1: Subopcode1
- COMMAND: USLOG

  OBJECT1: Subopcode1

SEL-CMD: Execute a preselection command

If the name or value of a parameter is longer than 8 characters, the abbreviation described in UTM-SAT administration is output in the log data field.

UTMUSER	Name of the UTM user who initiates the event.
UTMAPPL	Name of the active application.
UTMSUBC	SEL-CMD
COMMAND	Specification of the UTM-SAT administration command: MSATSEL or MSATPROT.
UTMREAS	Internal system return code.
UTMTAID	Transaction identification.
TACNAM	Transaction code of active program unit.
USER2	User or blank.

The fields DATNAM2, LTERM and OBJECT1 may be output as blank.

The following additional fields are defined, depending on COMMAND:

- MSATSEL (control preselection, command KDCMSAT SATSEL=...)

  A separate log data record is written for each name specified. Only one of the fields USER2, DATNAM2 or OBJECT1 is defined:

  USER2: UTM user as preselection object.

  DATNAM2: TAC as preselection object.

  OBJECT1: Event as preselection object.

  OBJECT2: Preselection value (NONE, SUCC, FAIL, BOTH or OFF).
- MSATPROT (Control SAT logging, command KDCMSAT SAT=...)

  OBJECT2: Logging switched on (ON) or off (OFF).

CHG-PROG: Exchange a load module

UTMAPPL	Name of the active UTM application.
UTMSUBC	CHG-PROG
UTMNAME	Name of the module to be exchanged.
UTMOBJ4	Load mode of the module to be exchanged.
UTMOBJ6	New module version.
UTMTAID	Transaction identification if UTMOBJ4=ON-CALL, otherwise zero.

For UTMOBJ4=ON-CALL, note that CHG-PROG is always logged as successful. The associated END-PU log data record indicates whether or not the exchange was successful. (In the event of an error, a PEND ER occurs with the corresponding KDCS return codes.) The initial loading is also logged as an exchange.

15.7 Sample programs

openUTM is shipped as standard with sample programs that make it easier for you to create applications. Some of these sample programs are explained in more detail below. The sample programs for administration can be found in the openUTM manual “Administering Applications”.

15.7.1 Sample programs for the sign-on service

Using the sample programs for the sign-on service, you can implement a simple sign-on service with FHS format output.

Both compiled objects and source programs (COBOL) are supplied. The new functions can thus be tried out quickly without programming. The source programs represent a programming template that can be used if you want to implement a sign-on service adapted to your own requirements.

Functions

The programs are suitable for all generation variants.

If required, the terminal user is requested to input sign-on data after the connection has been established. For this purpose, the sign-on service outputs a “welcome format” containing two input fields for the user ID and password. The password must be specified in printable form. Please remember when specifying the user ID and password that small letters are converted to capital letters.

If openUTM rejects the sign-on data, the sign-on service repeats the request for input. It then outputs the same format, though it now contains information on the rejection. After three failed attempts, the sign-on service is terminated.

If openUTM accepts the sign-on data, the sign-on service proceeds as follows:

- No service restart: A start format is output if one is generated; otherwise, a request is issued for input in line mode.
- Service restart: The screen restart of the open service is initiated.

English text is output on the terminal. The comments in the source program are also in English.

Components

The sample programs for the sign-on service are located in the library SYSLIB.UTM.070.EXAMPLE.

Element	LMS type	Meaning
SIGN1	S	COBOL source --> 1st program unit
SIGN2	S	COBOL source --> 2nd program unit
KDCSIGN1	R	Object module --> 1st program unit
KDCSIGN2	R	Object module --> 2nd program unit
FORSIGN	R	Screen format (' *' format)
FORSIGN	S	Addressing tool
FORSIGN	F	IFG source

Integration in a UTM application

To integrate the sample sign-on service in a UTM application, the KDCDEF generation statements must be extended as follows:

```
PROGRAM KDCSIGN1,COMP=ILCS
PROGRAM KDCSIGN2,COMP=ILCS
TAC     KDCSGNTC,PROGRAM=KDCSIGN1
TAC     TACSIGN2,PROGRAM=KDCSIGN2,CALL=NEXT
```

The TAC name TACSIGN2 is programmed. It is defined as a constant at the start of both program units and can thus be changed easily if required.

The standard primary working area must be at least 600 bytes, the communication area at least 2 bytes long (see MAX statement in the openUTM manual “Generating Applications”).

The linkage editor statement must be extended as follows:

```
INCLUDE-MODULES LIBRARY=$userid.SYSLIB.UTM.070.EXAMPLE ,ELEMENT=(KDCSIGN1, KDCSIGN2)
```

The screen format FORSIGN must be incorporated in the format library of the application. The format is suitable for terminals of types 8160, 9750, 9755 and 9763.

The program unit SIGN1 uses the COBOL85-specific statement EVALUATE.

15.7.2 Sample programs for a publish / subscribe server

These sample programs are intended to illustrate how to implement a simple publish and subscribe service in a UTM application.

Function

A user can subscribe to a service. That user then receives all messages published as of that time in their USER queue.
The possible commands for this service are:

- help: Get help text
- subscribe: Subscribe to messages
- unsubscribe: Cancel subscription to messages
- who: Output the names of the subscribers
- publish <*message*>: Publish a message

The service is provided by an asynchronous service with the TAC PUBSUBA which constantly listens for jobs at the TAC queue PUBSUBMQ. Users communicate with the service over the dialog service PUPSUBD. Job confirmations are sent to the USER queue of the user and can, for instance, be read using the dialog program UPDGET (see sample programs for asynchronous processing for a UPIC client). In addition, PU can be queried in the INIT of each program unit to establish whether messages are waiting in the user's queue.

The service need only be started once by calling the TAC PUBSUBA. The open asynchronous service is then retained throughout the entire duration of the application. It is transferred to the new application by KDCUPD when a new generation is performed.

If the asynchronous service terminates abnormally as the result of an error, the most recently processed job is placed in the dead letter queue.

Delivery

On BS2000 systems, source programs and object modules are supplied as members of the library SYSLIB.UTM.070.EXAMPLE.

Element	LMS type	Meaning
PUBSUBD.C	S	Issues a request to the publish/subscribe service, dialog program unit
PUBSUBA.C	S	Implements the publish/subscribe service, asynchronous program unit
PUBSUBD#LLM	R	Object module for PUBSUBD.C
PUBSUBA#LLM	R	Object module for PUBSUBA.C

UTM generation

The statements for the program units in the KDCDEF run are specified as comments in the individual source files. This also applies to the statement for the TAC queue "PUBSUBMQ".

At least one GSSB must be generated (MAX GSSBS), as the service uses the GSSB "PUBSUBGB" to manage the subscribers.

If the most recently processed job is to be placed in the dead letter queue after the service is cancelled, MAX REDELIVERY = (...,0) must be generated. Is this is not done, the job remains in the job queue PUBSUBMQ.

15.7.3 Sample program for moving messages from the dead letter queue selectively

Function

The dialog program moves all messages from the dead letter queue using a specified original destination and a specified new destination. This means that a total of 16 characters are expected, i.e either two TACs or two LPAP partners or two OSI-LPAP partners, depending on the type of the original target. The program confirms the number of messages moved.

Delivery

On BS2000 systems, source programs and object modules are supplied as members of the library SYSLIB.UTM.070.EXAMPLE.

Element	LMS type	Meaning
DADMMVS	S	Moves messages with a particular original destination from the dead letter queue, COBOL dialog program unit
DADMMVSC.C	S	Analog dialog program in C
DADMMVS	R	Object module for DADMMVS
DADMMVS#LLM	R	Object module for DADMMVSC.C

UTM generation

The statements for the program units in the KDCDEF run are specified as comments in the individual source files.

15.7.4 CPI-C sample programs

You will find the CPI-C sample programs in the library SYSLIB.UTM.070.XOPEN.

Element	LMS type	Meaning
KCPSAM1.C	S	C source (asynchronous part)
KCPSAM2.C	S	C source (synchronous part)
KCPSAM1#LLM	R	Object module (asynchronous part)
KCPSAM2#LLM	R	Object module (synchronous part)

15.7.5 Sample programs for asynchronous processing with UPIC clients

Three program units, namely UPDIAL, UPASYN and UPDGET, are supplied with openUTM for asynchronous processing with UPIC clients.

Functions

These three program units illustrate how to issue asynchronous requests from a UPIC client and how to receive asynchronous result information.

In this example, the asynchronous message is first sent to the USER queue of the user ID under which the UPIC client signed on. The message is then read by an interactive program unit and output on the client. The advantage of asynchronous processing is that the user at the UPIC client can enter a new request as soon as the current request is accepted and is not blocked until the request is completed.

These three programs have the following functions:

- UPDIAL reads an input message, sends it to the UPASYN program unit as an asynchronous request, and outputs a request confirmation on the client.
- UPASYN receives the message, waits 5 seconds to simulate complex processing, and writes the result in the USER queue of the user ID under which the UPIC client signed on.
- UPDGET waits (60 seconds) and reads the user ID of the service (if the dialog message is empty) or the user ID passed in the dialog message from the USER queue. This means that the UPDGET can run under a different user ID (e.g. without using a security user) from the UPDIAL and UPASYN services and fetch the message from the user queue of the user that started the UPDIAL service.

  If a dialog message that is not empty does not contain a valid user ID, UPDGET terminates with an error message. If no queue message is available, UPDGET is started again as soon as a message arrives or when the wait time has elapsed. If a queue message is available, it is sent to the UPIC client with MPUT and its own TAC is called again with PEND-RE in order to wait for the next message.

Components

The source programs and object modules are supplied as elements of the SYSLIB.UTM.070.EXAMPLE library. The sample programs should only be run in conjunction with a UPIC client program adjusted appropriately.

Integration in a UTM application

If the client is only to run with users generated with RESTART=NO, the procedure is as follows:

The UPIC client maintains two connections to the UTM application and signs on with the same user ID for openUTM. In the first thread, the client starts the interactive service UPDGET to read the asynchronous messages. In the second thread, the client starts the interactive service UPDIAL on explicit request, which then generates an asynchronous request for UPASYN.

If the client is also to run with users generated with RESTART=YES, the following procedure is possible:

The client only signs on as a security user in the second thread, and in the first thread signs on without explicitly specifying a security user. openUTM therefore allocates it a user permanently assigned to the connection. The first thread then also returns the name of the security user in the dialog message when UPDGET is started (and at every step in the dialog).

15.7.6 Sample programs for HTTP-Clients

The file SYSLIB.UTM.070.EXAMPLE contains four sample programs for HTTP clients as source and LLM.

More information about function and generation can be found in the sources. These are the following programs:

- HTTPECHO.C
- HTTPEXH.C
- HTTPEXT.C
- HTTPINF.C

15.8 Sample procedures

openUTM is supplied as standard with sample procedures that should simplify your work with openUTM. The procedures are designed as tools and templates that you can modify and extend in accordance with your requirements. The procedures contain comments in English.

These procedures have already been compiled (SYSJ members) so that they will also run in the basic configuration with SDF-P-BASYS. The source files are contained in SYSLIB.UTM.070.EXAMPLE.

The SYSPRC.UTM.070 library contains the following procedures:

Procedure	**Function**
BTRACE	Mix BCAM trace output files and evaluate with the KDCBTRC program
DUMP	Evaluate UTM dumps
FGGUSLOG	Create user log file as file generation group (FGG) and switch to the new user log file generation
GEN	Generate UTM application
LINK	Link UTM application program
MSGMOD	Create user-specific (message) modules
PAMSAM	Edit and sort data recorded by KDCMON
SLOG-FGG	Evaluate individual file generations of a SYSLOG-FGG
START-APPL-ENTER-PROC	Create an SDF procedure for starting an UTM application via ENTER-PROC
SHOW-ETPND	Display ETPND of a UTM module
START-BLS-APPLICATION	Start UTM production application with BLS
SYSLOG	Edit SYSLOG file
UPD	Transfer data to KDCFILE with KDCUPD

15.9 XS-support of UTM applications

This section contains a few particularities that you should take into consideration when linking and starting UTM applications that are to be loaded into the upper address space of a XS system and that are to run in 31-bit mode. You will need to be familiar with XS programming and the use of XS.

openUTM only supports applications that:

- are completely XS-compatible and run in 31-bit mode or
- are not XS-compatible and are completely loaded in the lower address space (i.e. below 16 Mbyte) and run in 24-bit mode.

A UTM application can therefore only be loaded into the upper address space (>16 Mbyte) and run in 31-bit mode when all the components of the UTM application program are XS-compatible.

> **CAUTION!**
>
> UTM applications in “mixed mode” (i.e. applications that switch between 24-bit and 31-bit address modes) are not supported by openUTM. This means that openUTM cannot guarantee that a UTM application will run properly if a program unit that runs in 31-bit mode will dynamically load modules in 24-bit and switch to the appropriate address mode by itself when entering these modules, for example.

Compiling and linking

The following rules are to be observed when compiling and linking an XS-compatible UTM application:

- All program units must be compiled with the attributes AMODE=ANY (addressing mode) and RMODE = ANY (resident mode).
- When linking the UTM application, the binder checks the AMODE and RMODE attributes for all program units and sets a pseudo-RMODE for the object module of the UTM application created. The BINDER bases its setting on the “weakest” component, i.e. the module is only assigned the attribute RMODE=ANY when all components have the RMODE=ANY attribute. If a component was compiled with RMODE=24, the module is assigned RMODE=24.

  The AMODE attribute is determined by the program section (CSECT) that contains the entry point of the object module.
  You will find more information in the BS2000 manual “Dynamic Binder Loader / Starter”.

Particularities when starting a UTM application

Whether the UTM application is loaded into the upper or lower address space depends on the UTM application program itself and on the value of the PROGRAM-MODE parameter that you specified when calling the START-EXECUTABLE-PROGRAM command.

- For PROGRAM-MODE=24 the application is loaded into the lower address space and the 24-bit mode is set.
- For PROGRAM-MODE=ANY:
  Whether the UTM application is loaded into the upper or lower address space and which addressing mode is set depends on the attributes AMODE and RMODE of the load module (see section “Compiling and linking”).

  If the Binder-Loader System (BLS) detects at the start of the application that all components of the UTM application loaded at the start of the application are XS-compatible, then the UTM application is loaded into the upper address space.

If openUTM is to dynamically load a non-XS-compatible module in the start phase of an application loaded in the upper address space, then openUTM aborts the start procedure with an appropriate error message.

You should also note that no 24_bit modules (ONCALL) may be loaded dynamically during live operation by an application program that runs in the upper address space.

To ensure that a non-XS-compatible UTM application is loaded into the lower address space and runs in 24-bit mode, you need to add a MODIFY-SYMBOL-ATTRIBUTES statement with AMODE=24 when linking the UTM application.

Memory allocation of UTM applications

openUTM creates the application-specific tables and data areas (KAA, KTA, slots and UTM cache) in class 5 memory in the upper address space. Address space is therefore not taken away from UTM applications that run in the lower address space.

By means of MAX CACHE generation, the UTM cache can also be stored in one or more data spaces.

The KDCDEF, KDCDUMP and KDCUPD tools

The UTM tools KDCDEF, KDCDUMP and KDCUPD only run in the upper address space (> 16 MByte).

16 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.

17 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

18 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*