English

Fujitsu Software BS2000

openUTM V7.0

Generating Applications

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

Generating Applications

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

The openUTM manual “Generating Applications” is designed for use by application planners and developers as well as operators of UTM applications.

This manual describes how to define the configuration for a UTM application using the UTM tool KDCDEF and how to create the KDCFILE. Chapter 5 also goes into more detail about the generation of selected objects and functions of the application.

Additional topics include the dynamic configuration of an application and the updating of the KDCFILE using the tool KDCUPD.

To understand this manual you will need to be familiar with the operating system.

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 Introduction to the generation procedure

Alongside the program units that provide the services and the formats (for formatted operation on BS2000 systems) you must create the following application components for a UTM application:

- KDCFILE configuration file
  The KDCFILE contains the configuration of your application. openUTM stores all the administrative data required to operate the application in the KDCFILE and reserves areas for the user data and for transaction management. When operating the application, all tasks and work processes of the application access the KDCFILE. This manual describes how to create the KDCFILE.
- KDCROOT main routine
  The program units you have created run under the control of KDCROOT. The ROOT tables are used as the basis of the main routine KDCROOT.
  The ROOT tables contain application-specific configuration data that is required by the main routine KDCROOT. When operating an application, the main routine KDCROOT establishes the connection from openUTM to the program units, the database and on BS2000 systems to the formatting system.
  This manual also describes how to create the sources for the ROOT tables.

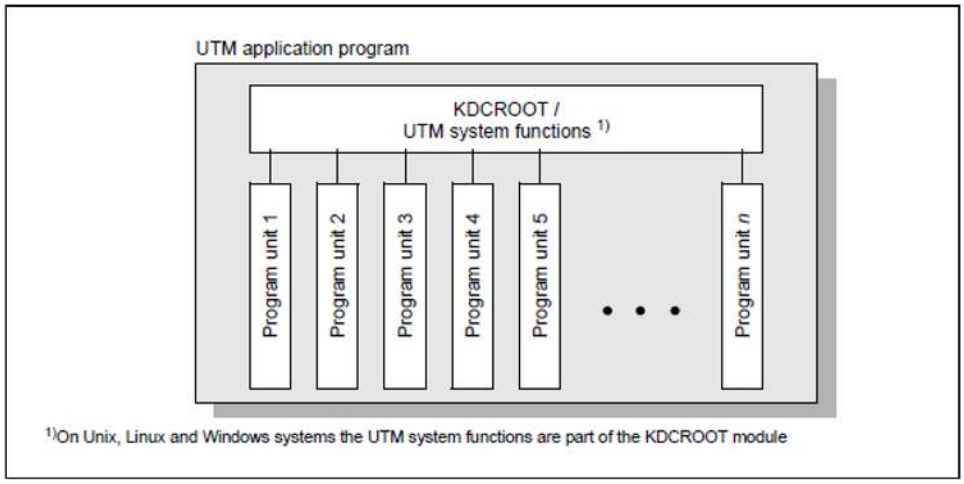


Figure 1: Structure of the UTM application program

In order to create the ROOT table source and the KDCFILE, you must first define the configuration of application. This entire procedure is known as “generation”. To allow you to configure and generate the KDCFILE and the ROOT table sources, openUTM provides the KDCDEF generation tool.

UTM applications can be operated in the form of a **standalone application** (one host) and, on Unix, Liniux and Windows systems, also in the form of a **UTM cluster application** (on more than one host).

The KDCDEF generation tool is described in detail in section "The KDCDEF generation tool".

Information on the KDCFILE can be found in section "The KDCFILE".

The information on generation contained in this section "Introduction to thegeneration procedure" applies both to standalone UTM applications and to UTM cluster applications. You will find additional information on generating UTM cluster applications in section "Notes on generating a UTM cluster application on Unix, Linux and Windows systems".

You must create the application program using KDCROOT, user program units, interfaces and other application components like the UTM system modules, the runtime systems of the programming languages, database connection modules etc.

More information about creating application programs using ROOT tables and application components can be found in the corresponding openUTM manual “Using UTM Applications”.

Information about creating application program units can be found in the openUTM manuals “Programming Applications with KDCS” and “Creating Applications with X/Open Interfaces”.

Information about creating formats on BS2000 systems can be found in the manuals for FHS.

2.1 Configuring the UTM application

To execute the application program, you must define the following information for example:

- the application properties
- the UTM user IDs and data access control
- the properties of clients and printers
- the properties of partner applications (server applications)
- the properties of services, i.e. of transaction codes and program units
- message queues (user, TAC and temporary queues)
- the structure of the application (subdivision into load modules for use with BLS, shared objects or DLLs)
- reserved locations in UTM object tables for dynamic configuration

These properties combine to form the configuration, and are defined using the KDCDEF control statements. The KDCDEF control statements serve as input for the generation tool KDCDEF.
The KDCDEF control statements are listed in accordance with their function group starting in section "Creating the ROOT table source, the KDCFILE and UTM cluster files".

The KDCFILE administrative file is used to store all configuration information and thus all administrative data required to run the application.

2.2 Generating application components - result of the KDCDEF run

You can generate the KDCFILE and the ROOT table sources in a single KDCDEF run or in separate KDCDEF runs. The KDCDEF statement OPTION allows you to define the generation objects to be created by KDCDEF:

```
OPTION...,GEN={ KDCFILE | ROOTSRC | NO | ALL }
```

For UTM cluster applications on Unix, Linux and Windows systems, there is the additional option CLUSTER, see "OPTION - manage the KDCDEF run".

The name of the ROOT tables is defined using the ROOT statement.

```
ROOT rootname
```

On BS2000 systems *rootname* is the name of the ROOT table module.

On Unix, Linux and Windows systems *rootname* is a name component of the ROOT table source (ROOTSRC).

KDCDEF reads the control statements from standard input or from a file.

On BS2000 systems standard input means SYSDTA (with the SDF command ASSIGN-SYSDTA you can assign SYSDTA to a SAM or ISAM file, a library member of type S, a PLAM library, or *SYSCMD, for example)

On Unix, Linux and Windows systems standard input means *stdin* (i.e. from the Unix or Windows command level).

You will find a detailed description of how to start KDCDEF and pass the control statements to KDCDEF in section "Calling KDCDEF and entering the control statements".

All KDCDEF statements are subjected to syntax and plausibility checks. If KDCDEF does not detect any serious errors in this process, the files listed in figure 2 are created for a standalone UTM application.

Figure 17 in chapter "UTM cluster files" shows what files are created when you generate a UTM cluster application.

> ¡ Even if the OPTION statement is used in a KDCDEF run to cause only part of the configuration to be (newly) created, you nevertheless specify the statements for the entire configuration for every generation run. Only then is KDCDEF able to check the completeness and consistency of the generation statements.
>
> KDCDEF always performs plausibility checks for all statements. If, for instance, only a ROOT source is to be generated in a KDCDEF run, KDCDEF also checks the statements that only affect the KDCFILE.
>
> This complete check allows inconsistencies that arise on creating the ROOT table module and the KDCFILE that would otherwise only be detected when the application is started to be identified early and consequential errors to be avoided.

The following figure shows what files are created when you define a standalone UTM application.

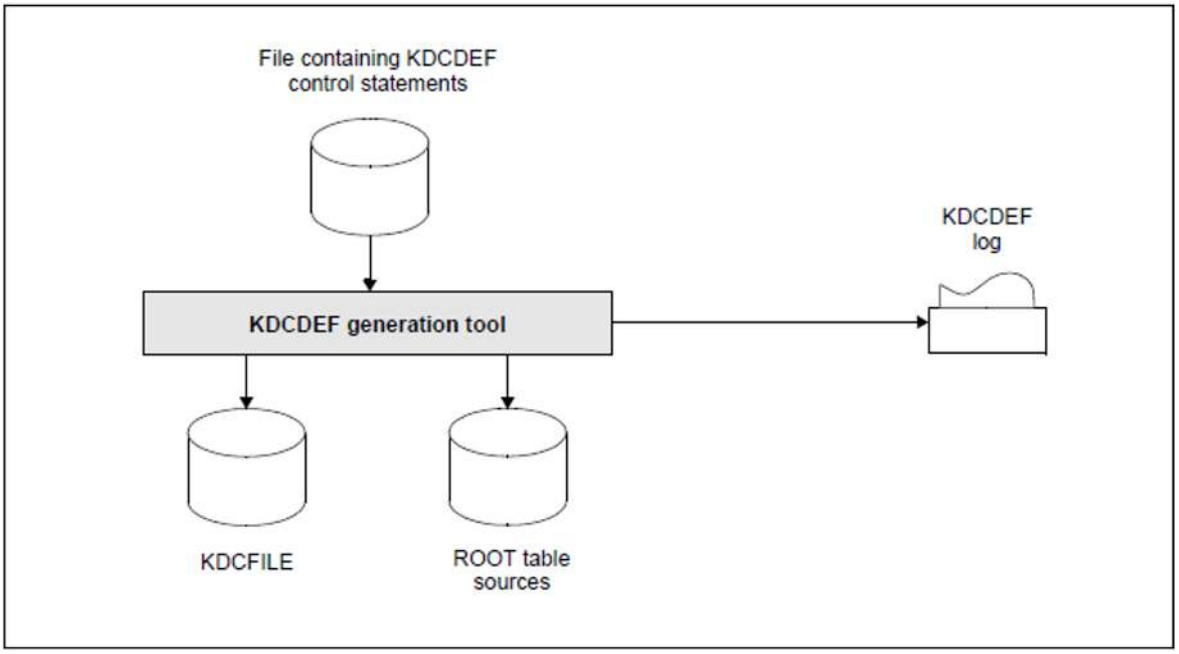


Figure 2: The result of the KDCDEF run (with OPTION ...,GEN=ALL) for a standalone UTM application.

KDCDEF statements for a minimal configuration

You must pass at least the following control statements to KDCDEF before you can run your UTM application.

You must execute additional KDCDEF statements for distributed processing, connecting specific clients and printers, etc. You will find more information on this subject in the sections "Generating applications for distributed processing", "Connecting clients to the application" and "Generating printers (on BS2000, Unix and Linux systems)".

The lines beginning with '*' are comments.

Minimal configuration for BS2000 systems:

```
*******************************************************************************
*
* Specify which part of the application program is to be created by KDCDEF
*
*******************************************************************************
*

OPTION GEN=...
```

```
*******************************************************************************
*
* Specify the name of the Root table
*
*******************************************************************************
*

ROOT applroot
```

```
*******************************************************************************
*
* Specify application parameters
*
*******************************************************************************
*
* Application name with which communication partners
*    can address the application

MAX APPLINAME= sample

 * Specify the base directory of the application.

MAX KDCFILE= filebase

* Define the maximum number of process of the UTM application


MAX TASKS=2
```

```
*******************************************************************************
*
* optional: Generate the database system (ORACLE in the example)
*
*******************************************************************************
*

DATABASE TYPE=XA
```

```
*******************************************************************************
*
* optional: Specify the formatting system used
*
*******************************************************************************
*
* the statement FORMSYS must only be specified if your UTM application
* is to run in formatted mode

FORMSYS TYPE=FHS
```

```
*****************************************************************************
*
* Connection points (LTERM partners) for clients/TS applications
*
*****************************************************************************
*
* For example, generate open LTERM pools so that clients/TS applications
* can connect to the application

* LTERM pools for the various types of client ------------------------------ (1)

TPOOL LTERM= client, NUMBER=..., PTYPE=*ANY, PRONAM=*ANY
BCAMAPPL upicappl, T-PROT=ISO
TPOOL LTERM= upic, NUMBER=..., PTYPE=UPIC-R, BCAMAPPL= upicappl, PRONAM=*ANY
TPOOL LTERM= appli, NUMBER=..., PTYPE=APPLI, PRONAM=*ANY
BCAMAPPL sockappl, T-PROT=SOCKET, LISTENER-PORT=number
TPOOL LTERM= socket, NUMBER=..., PTYPE=SOCKET, BCAMAPPL=sockappl, PRONAM=*ANY
```

```
*****************************************************************************
*
* Generate services
*
*****************************************************************************
*
* Some own program units that initiate services and generate the
* corresponding transaction codes
* (for COMP=... enter the compiler used or the runtime system, mostly „ILCS“,)

PROGRAM= userpu ,COMP=...
TAC usertc ,PROGRAM= userpu. ...----------------------------------------- (2)
```

```
*****************************************************************************
*
* Administration
*
*****************************************************************************
*

* Administration program KDCADM

PROGRAM KDCADM,COMP=ILCS )

* Generate administration command KDCSHUT so that the application
* can always be terminated normally

TAC KDCSHUT,PROGRAM=KDCADM ... ------------------------------------------- (3)

* In applications with user IDs:
* user ID for the administrator

USER admin ,PERMIT=ADMIN,PASS=....

* If administration will be done via WinAdmin/WebAdmin,
* then you need to submit the following TAC and PROGRAM statements
* and generate a connection for the UPIC client (an LTERM pool in this case)
* Furthermore, you should then generate the admin user ID with administration
* authorization and without the restart property or generate your own
* user ID with administration authorization and without the restart property
* for administration using WinAdmin/WebAdmin.

* Administration program KDCWADMI

PROGRAM KDCWADMI,COMP=..
TAC KDCWADMI,PROGRAM=KDCWADMI,CALL=BOTH,ADMIN=Y
TPOOL LTERM=WADM,PTYPE=UPIC-R, PRONAM=*ANY, NUMBER=1 ------------------------(5)
```

```
*****************************************************************************
*
*optional: Reserve space in the table for dynamic administration
*
*****************************************************************************
*

RESERVE OBJECT=...,NUMBER=... ------------------------------------------- (4)

END
```

Remarks	
(1)	For each of the client types (terminal, UPIC client, TS application) that are to connect to the application, you must generate a separate LTERM pool. For terminals, a single LTERM pool is sufficient - depending on the type of terminals that are to sign in to the application. You can also generate the LTERM pools so that all clients of a particular type can log in - regardless of the computer on which they are located. You can also implement client connections with the help of the LTERM/PTERM statements. In particular, you must use LTERM/PTERM statements if the UTM application itself establishes connections to clients (e. g. TS applications) or if a printer is to be generated. For UPIC clients, HTTP clients and TS socket applications a separate BCAMAPPL statement with T-PROT=ISO or (SOCKET, ...) is required.
(2)	You can also assign several transaction codes to a program unit if the program unit performs several different services.
(3)	You can generate all administration commands that you will want to use in operation using additional TAC statements. If you want to use your own administration programs for administration purposes, then you must generate these programs with the corresponding PROGRAM and TAC statements.
(4)	You can add additional objects to the application configuration during live operation with the help of the administration (see the openUTM manual “Administering Applications”). You will need to create space in the tables in the KDCFILE for these objects in the KDCDEF generation.
(5)	The connection for a WinAdmin or WebAdmin client can also be generated with PTERM-/LTERM statements instead of with a TPOOL statement - e.g. with a privileged LTERM, see openUTM manual “Administering Applications”.

Minimal configuration for Unix, Linux and Windows systems

```
*******************************************************************************
*
* Specify which part of the application program is to be created by KDCDEF
*
*******************************************************************************
*

OPTION GEN=...
```

```
*******************************************************************************
*
* Specify the name of the Root table
*
*******************************************************************************
*

ROOT applroot
```

```
*******************************************************************************
*
* Specify application parameters
*
*******************************************************************************
*
* Application name with which communication partners
*        can address the application

MAX APPLINAME= sample

* Specify the base directory of the application.
* This directory is the directory in which the KDCFILE is stored,
* amongst other things.

MAX KDCFILE= filebase

* Specify the key for the shared memory area

MAX CACHESHMKEY=..,IPCSHMKEY=...,KAASHMKEY=...
                [,OSISHMKEY=...,XAPTPSHMKEY=...] --------------------------- (1)

Define the semaphore key for the global application semaphore
MAX SEMARRAY= number , number1

* Define the maximum number of process of the UTM application.

MAX TASKS=2
```

```
*****************************************************************************
*
* optional: Generate the database system (in the example ORACLE)
*
*****************************************************************************
*

RMXA XASWITCH=xaoswd,SPEC=C
```

```
*****************************************************************************
*
* Connection points (LTERM partners) for clients/TS applications
*
*****************************************************************************
*

* For example, generate open LTERM pools so that clients/TS applications
* can connect to the application

* LTERM pools for the various client types -------------------------------- (2)

TPOOL LTERM=clientr, PTYPE=UPIC-R, NUMBER=...
TPOOL LTERM=clientl, PTYPE=UPIC-L, NUMBER=...
TPOOL LTERM=appli, PTYPE=APPLI, NUMBER=...
BCAMAPPL aockappl, T-PROT=SOCKET, LISTENER-PORT=number
TPOOL LTERM=socket, PTYPE=SOCKET, BCAMAPPL=sockappl, NUMBER=...
TPOOL LTERM=term, PTYPE=TTY, NUMBER=...
```

```
*****************************************************************************
*
* Generate services
*
*****************************************************************************
*

* Some own program units that initiate services and generate the

* corresponding transaction codes (for COMP=... enter the compiler used)

PROGRAM= userpu ,COMP=...
TAC usertc ,PROGRAM= userpu. ...---------------------------------------- (3)
```

```
*****************************************************************************
*
* Administration
*
*****************************************************************************
*

* Administration program KDCADM

PROGRAM KDCADM,COMP=C

* Generate administration command KDCSHUT so that the application
* can always be terminated normally

TAC KDCSHUT,PROGRAM=KDCADM ...------------------------------------------- (4)

* In applications with user IDs: user ID for the administrator

USER admin ,PERMIT=ADMIN,PASS=....

* If administration will be done via WinAdmin/WebAdmin,
* then you need to submit the following TAC and PROGRAM statements
* and generate a connection for the UPIC client (an LTERM pool in this case)

* Administration program KDCWADMI

PROGRAM KDCWADMI,COMP=C
TAC KDCWADMI,PROGRAM=KDCWADMI,CALL=BOTH,ADMIN=Y
TPOOL LTERM=WADM,PTYPE=UPIC-R, NUMBER=1 ---------------------------------- (6)
```

```
*****************************************************************************
*
* optional: Reserve space in the table for dynamic administration
*
*****************************************************************************
*
RESERVE OBJECT=...,NUMBER=... ------------------------------------------- (5)

END
```

Remarks	
(1)	You only need to specify the shared memory key OSISHMKEY= and XAPTPSHMKEY= if you generate objects for communication via OSI TP. The other shared memory areas are needed by every UTM application, running on Unix, Linux or Windows systems.
(2)	On Unix, Linux or Windows systems you must generate a separate LTERM pool for each type of client (terminal, UPIC client, TS application) that is to be able to connect to the application. You can generate the LTERM pools so that all clients of a particular type are able to sign on - regardless of the computer on which they are located. You can also configure client connections using the LTERM/PTERM statements. In particular, you must use LTERM/PTERM statements if the UTM application itself establishes connections to clients (e.g. TS applications) or if a printer is to be generated on Unix or Linux systems. For HTTP clients and TS socket applications a separate BCAMAPPL staement with T-PROT=(SOCKET, ..) is required.
(3)	You can also assign several transaction codes to a program unit if the program unit performs several different services.
(4)	You can generate all administration commands that you will want to use in operation using additional TAC statements. If you want to use your own administration programs for administration purposes, then you must generate these programs with the corresponding PROGRAM and TAC statements.
(5)	You can add additional objects to the application configuration during live operation with the help of the administration (see the openUTM manual “Administering Applications”). You will need to create space in the tables in the KDCFILE for these objects in the KDCDEF generation.
(6)	The connection for a WinAdmin or WebAdmin client can also be generated with PTERM-/LTERM statements instead of with a TPOOL statement - e.g. with a privileged LTERM, see openUTM manual “Administering Applications”.

Regenerating existing UTM applications

If you want to generate a new ROOT table source and/or a new KDCFILE for an existing application (i.e. KDCROOT and KDCFILE already exist), then you must note the following:

You must enter the information on objects that are entered dynamically in the KDCFILE during operation or whose properties have been changed in the new KDCFILE. The “inverse KDCDEF“ function is provided for this purpose. With this function you create the control statements from the configuration information of the current KDCFILE that can be used immediately. You will need to call the CREATE-CONTROL-STATEMENTS control statement in the KDCDEF run in order to do this.

Via the UTM administration you can also execute the inverse KDCDEF run while the application is running.

You will find more information on the “inverse KDCDEF“ function in section "Inverse KDCDEF".

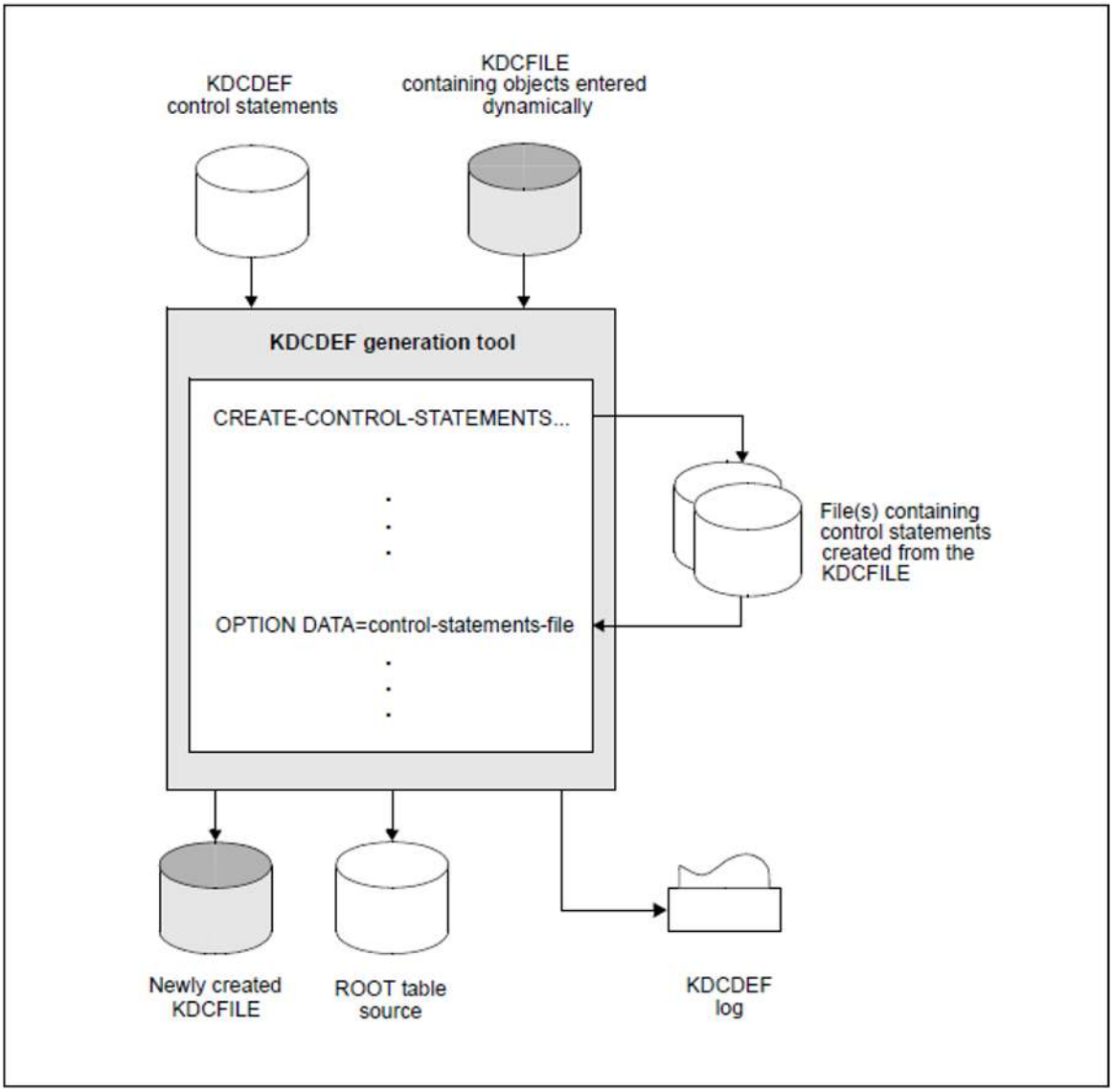


Figure 3: KDCDEF run with inverse KDCDEF

2.3 The KDCFILE

The KDCFILE contains all data required to run a UTM application. It is shared by all application processes during runtime. In its most basic form, the KDCFILE consists of a single file (a PAM file on BS2000 systems). The KDCFILE can also be distributed over several files. For security reasons, it can be duplicated.

The KDCFILE is logically divided into three areas:

- **Administrative data**, see "Administrative data"
- **Page pool**, see "Page pool"
- **Restart area**, see "Restart area"

KDCDEF generation

The KDCFILE is generated during the KDCDEF run by specifying

```
OPTION...,GEN=KDCFILE or GEN=ALL
```

in the KDCDEF statement.

The following characteristics of the KDCFILE must be specified at the KDCDEF generation:

- Data block size
  Each area within the KDCFILE is organized in units of either 2K, 4K or 8K. These units are known as UTM pages. You can define the block size of a UTM page using the following control statement:

  ```
  MAX...,BLKSIZE={ 2K | 4K | 8K }
  ```

  Whether a block size of 2K, 4K or 8K is to be favored depends on the sizes of the data areas (GSSB, LSSB, etc.) and the lengths of the messages that your program uses. See also "Page pool" for more information.

- Base name of the KDCFILE
  You specify the base name (called the *filebase* in the following) and single or dual-file operation of the KDCFILE with:

```
MAX..., KDCFILE={ filebase [, SINGLE | DOUBLE ] }
```

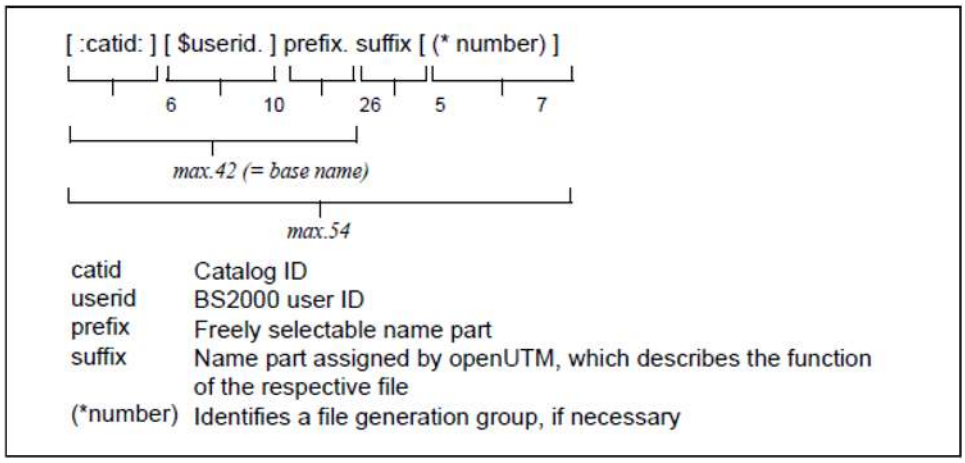
  In the case of dual-file operation, the contents of both KDCFILE files are always identical. If one of the files is corrupted, it can be restored by simply copying the other file.

  The name specified in *filebase* is also the base name of additional files and file generations of the application (for example, the system and user log file). *filebase* is therefore the base name of the application.

  The significance of the base name *filebase* is different for each of the various platforms:

  BS2000 systems:

  The full names of the files derived from *filebase* have the following format:



  The total length of the file name must not exceed 54 characters. The base name *filebase* can be up to 42 characters in length, including *userid* and *catid*. If no *catid* or *userid* is specified when defining the base name (*filebase=prefix)*, the lengths of these fields must still be taken into consideration when determining the total length of the file name.

  If file generation groups (i.e. USLOG files, SYSLOG generation group) are not used in the application, *prefix* can be up to 33 characters in length. Otherwise, *prefix* must not exceed 26 characters.

  The KDCDEF generation tool then creates the following files:

  - *filebase*.KDCA in the case of single-file operation (SINGLE)
  - *filebase*.KDCA and *filebase*.KDCB in the case of dual-file operation (DOUBLE)

  When splitting the KDCFILE, additional files are created, see "Splitting the KDCFILE".

  Unix, Linux and Windows systems:

  filebase specifies the name of the base directory in which the KDCFILE is stored.

  The KDCFILE is created by KDCDEF under the *filebase* directory, where *filebase* is the fully qualified name of a directory which must be created **before** the KDCDEF run.

  The KDCDEF generation tool creates the following files in the *filebase* file directory:

  - KDCA in the case of single-file operation (SINGLE)
  - KDCA and KDCB in the case of dual-file operation (DOUBLE)

- Size of the page pool
  You can define the size of the page pool using the following control statement:

  ```
  MAX...,PGPOOL=( number, warnlevel1, warnlevel2 )
  ```

  Further information can be found in section "Page pool".
- Size of the restart area
  You can define the size of the buffer and the restart area using the following control statement:

  ```
  MAX ...,RECBUF=( number,length )
  ```

  Further information can be found in section "Restart area".

During generation, the page pool and the restart area can be distributed over several files. Further information can be found in section "Splitting the KDCFILE".

Data security - dual-file operation of the KDCFILE

For security reasons, it may make sense to duplicate the KDCFILE (dual-file operation). If one of the files is destroyed, then you can continue working with the other KDCFILE without losing data.

Dual-file operation of the KDCFILE does not have any significant effect on I/O times (these are certainly not doubled), and therefore does not reduce performance.

With dual operation of the KDCFILE, it makes sense to store the two files on different volumes (disks). If one of the volumes is physically damaged, this ensures that a viable copy is still available.

BS2000 systems:

You can create the files on the desired volumes by issuing appropriate /CREATE-FILE commands before the KDCDEF run or by copying the files after the KDCDEF run. When generating the application, you can also use the CATID parameter of the MAX statement to assign different CATIDs to the two files.

A copy of the KDCFILE is maintained when you specify
`MAX KDCFILE=(....,DOUBLE)`
in the KDCDEF generation.

Unix and Linux systems:

On Unix and Linux systems you can store the two KDCFILEs on different disks. This is generally only possible with the help of symbolic links (ln -s) to raw devices or across different file systems across different file systems. In this manner, you have a copy of the file even if one of the two disks is physically destroyed.

A copy of the KDCFILE is maintained when you specify
`MAX KDCFILE=(....,DOUBLE)`
in the KDCDEF generation.

Windows systems:

On Windows systems you can additional increase data security with the operating resources already available. For example, you can use single-file operation for the KDCFILE (`MAX KDCFILE=(....,SINGLE)`) and create a mirrored image of the disk on which the KDCFILE is stored on another disk. During operation, all changes to the KDCFILE are also made on the mirror disk. Even if one of the disks is physical destroyed, you can still continue working with the other hard disk without losing data.

2.3.1 Administrative data

The administrative data area contains configuration information, such as application runtime parameters, lists of all objects that can be addressed by name, administrative data on the page pool and restart area, and tables of user IDs, clients, LTERM partners, transaction codes, key and lock codes, and function keys.

All tasks and work processes of the application work with the administrative data and use the application to exchange information.

The administrative data itself is initialized using the KDCDEF generation tool. When starting the application, it is loaded into a shared memory, which can then be accessed by all tasks/work processes of the application.

On BS2000 systems this memory is located in a Common Memory Pool.

On Unix, Linux and Windows systems the administrative data is put into a shared memory segment.

In a UTM-S application, openUTM writes the administrative data (including any modifications made in the meantime) back to the KDCFILE at certain intervals (Periodic Write). This also occurs at the end of the application run. This version of the administrative data then forms the basis for the next application run.

In a UTM-F application, openUTM writes only certain modified administrative data back to the KDCFILE, e.g. changed user passwords and configuration data incorporated by means of dynamic administration.

2.3.2 Page pool

The page pool stores user data created during the application run. This includes:

- LSSBs, GSSBs, TLS blocks, and ULS blocks
- message queues, i.e. asynchronous messages (including time-driven messages) for clients, asynchronous services and service-controlled queues, and the dead letter queue, among other items
- buffered user log records (USLOG)
- service data (KB program area, last dialog message, etc.)
- dialog messages buffered after input as a result of TAC class or priority control
- output messages to clients

A number of specific characteristics apply to UTM cluster applications on Unix, Linux or Windows systems, see "Notes on generating a UTM cluster application".

The active UTM application accesses the page pool via the UTM cache, see KDCDEF generation, MAX statement, CACHESIZE operand in section "MAX - define UTM application parameters".

You can define the size of the page pool (number of UTM pages) during KDCDEF generation using the MAX statement:

```
MAX...,PGPOOL=( number, warnlevel1, warnlevel2 )
```

number	Size of the page pool in UTM pages
warnlevel1	The first warning is output when the percentage utilization of the page pool reaches the value specified here
warnlevel2	The second warning is output when the percentage utilization of the page pool reaches the value specified here

The current utilization of the page pool can be queried by the administration, e.g. using the KDCINF PAGEPOOL command (see openUTM manual “Administering Applications”) or by means of WinAdmin or WebAdmin.

The utility program KDCUPD provides another way of obtaining more detailed information about the type of the data stored in the page pool. It is possible to display the number of pages used for each application object, e.g. for each user. You will find more information on this subject in section "The tool KDCUPD – updating the KDCFILE".

Estimating the necessary size of the page pool

Once the page pool size has been defined, it cannot be modified while the application is running. When designing a UTM application, it is therefore necessary that you estimate the page pool size which will be required during runtime. The page pool size cannot be modified until after the application has been terminated. This involves regenerating the KDCFILE using the KDCDEF generation tool, whereby the existing user data is copied then from the old KDCFILE to the new KDCFILE using the KDCUPD tool. Further information can be found in section "The tool KDCUPD – updating the KDCFILE".

When estimating the required page pool size, you must examine the behavior of the program units, identify which data areas are stored in the page pool, and determine their size. The following must also be noted:

- The page pool is divided into UTM pages:
  a UTM page is either 2KB, 4KB or 8KB in length, depending on the value of the BLKSIZE= parameter in the MAX statement.
- The following applies for the data areas GSSB, LSSB, TLS, and ULS:each of these data areas begins on a new UTM page. Of each UTM page, 1994 bytes (in the case of a 2KB UTM page), 4042 bytes (in the case of a 4KB UTM page) or 8138 bytes (in the case of a 8KB UTM page) are available for user data. The remaining space is reserved by openUTM.

  openUTM offers the option of compressing the user data of these areas, see the DATA-COMPRESSION parameter of the MAX statement. This can reduce the number of UTM pages required.
- The following applies to asynchronous messages:
  Each message begins on a new UTM page. Of the first UTM page of a message, at least 1914 bytes (in the case of a 2KB UTM page), 3962 bytes (in the case of a 4KB UTM page) or 8058 bytes (in the case of a 8KB UTM page) are available for user data. Of each follow-up page occupied by the message, at least 2030 bytes (in the case of a 2KB UTM page), 4078 bytes (in the case of a 4KB UTM page) or 8174 bytes (in the case of a 8KB UTM page) are available for user data.

  In future versions of openUTM, it is possible that less space will be provided for user data on each UTM page. When programming, therefore, you should ensure that all the available space is not exhausted.
- If an existing area is modified, openUTM stores the new area up to the end of the transaction at another location in the page pool. The area is thus temporarily duplicated.

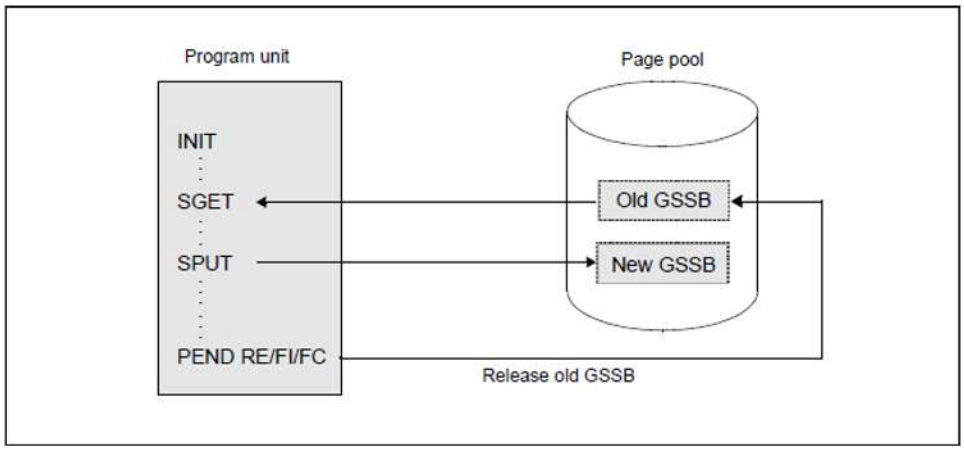


Figure 4: Dual-file operation of changed areas in the page pool

> You must also allow for the volume of FPUT and LPUT messages. Make sure that the page pool is not too small.

Page pool overflow warning

While the application is running, it is vital that the page pool does not become full as it is also required to back up the dialog messages. For this purpose, openUTM provides the following protective measures:

- There are two warning levels (percentages), which can be set during generation. If utilization of the page pool exceeds or falls short of these values, openUTM outputs UTM message K040 or K041, so that the user can respond with a MSGTAC routine.
- Local asynchronous messages and LPUT calls to write records to the USLOG file are rejected if utilization of the page pool has reached warning level 2.

- Asynchronous messages from a partner application via LU6.1 or OSI TP are rejected if utilization of the page pool has reached warning level 2. The connection is cleared. When communicating via OSI TP, the message `K119 OSI-TP error information` with the insert `DIA3=21` is output in both applications. The queued message is resent to the partner application at regular intervals determined by the value in MAX CONRTIME.
- An asynchronous job issued by a terminal or TS application is reject with UTM message K101 if utilization of the page pool has reached warning level 2.

2.3.3 Restart area

KDCS calls in a program unit will result in modifications to the administrative data.
openUTM collects information on all changes made within a transaction - i.e. from the first INIT call to the end of the transaction - in a process-specific storage area. On BS2000 systems this is a buffer in class 5 memory.

In a UTM-S application, openUTM uses the information in this buffer to create a restart data record at the end of the transaction. It then writes this data record to the restart area of the KDCFILE. The data record describes the modifications to the administrative data which were made as a result of the transaction. In the case of a warm start, it is used by openUTM to trace the effect of the transaction. The size of the restart area determines the interval at which modifications to the configuration data must be transferred to the administrative data area of the KDCFILE.

In a UTM-F application, restart data records are written only for transactions in which passwords were changed or in which administrative data was modified by means of dynamic configuration.

The data records in the restart area are combined by openUTM, i.e. a UTM page in this area generally contains several data records.

During KDCDEF generation, you can define the size of the buffer and the restart area using the following statement:

```
MAX ...,RECBUF=( number,length )
```

number Size of the restart area for each process in the KDCFILE, specified in UTM pages

length Size of the buffer for each process in the main memory, specified in bytes

Setting the *length* parameter

The *length* parameter lets you reserve for each process a buffer area that is *length* bytes long in main memory. openUTM uses this area to buffer changes to administrative data while a transaction is still open and can therefore still be rolled back.

For *length*, you must calculate the space requirements of the application’s transactions in the buffer using default values:

- In addition to the basic requirement of 40 bytes per transaction, you must also allow for the following:
  - up to 50 bytes per KDCS call, but 80 bytes per MCOM call.
  - up to 300 bytes per ADMI call.
- In the case of distributed processing, the following additional requirements must be taken into consideration:
  - 300 bytes per LU6.1 communication partner
  - 200 bytes per OSI TP partner

- In the case of asynchronous administration by means of an FPUT call, please note that all FPUT NT calls from a program unit to the same administration TAC are processed by the UTM administration program in a single transaction. The individual administration commands require the following buffer space, which must be taken into consideration in *length*:
  - 0 bytes for each KDCHELP and KDCINF administration command
  - for all other administration command others
    300 bytes on BS2000 systems
    360 bytes on Unix, Linux and Windows systems

  > ¡ On 64-bit platforms you will need to double the memory requirement.

  If RECBUF=*length* is generated too small, i.e. if the buffer is not large enough for a transaction, openUTM rejects a KDCS call or rolls back the transaction or cancel the transaction abnormally.

Setting the *number* parameter

The *number* parameter lets you reserve for each process a buffer area whose size is *number* UTM pages in the KDCFILE. openUTM uses this area to buffer changes to the administrative data of completed transactions until the changed data is written to the KDCFILE in the next periodic write. A UTM page generally contains several data records, since these generally occupy only slightly less space than the corresponding information in the buffer.
If *number* is defined too low, KDCDEF automatically increases this to the minimum value.

When combined with the restart records, the administrative data in the KDCFILE always represents the last valid state of the application. When using UTM-S, openUTM automatically updates the administrative data in the KDCFILE (Periodic Write) before a restart area becomes full during runtime. All pages containing administrative data in which modifications have been made are written to the KDCFILE parallel to the transactions currently active. All data records written previously to the restart areas thus become obsolete.

> ¡ If the restart area is large, the administrative data in the KDCFILE is updated less frequently during runtime. When performing a warm start following termination of the application, however, large quantities of data records from the restart areas must be incorporated in the KDCFILE, i.e. the warm start takes longer. The opposite applies if the restart area is small: since administrative data is updated frequently during runtime, the time required for warm start is reduced but impact slightly on the executing application.

number should be set such that the space available in the restart area is at least a multiple of the buffer generated using the *length* parameter.

In a UTM-F application, the volume of administrative data written back to the KDCFILE is much less, e.g. changed user passwords and generation data modified by means of dynamic configuration. The *number* parameter can therefore be set lower for UTM-F applications.

2.3.4 Creating a new KDCFILE during operation

To minimize the downtime for a UTM application during a new generation, it is also possible to create a new KDCFILE for an application while the application is running. However, you must bear the following in mind:

BS2000 systems:

The base name of the new KDCFILE consists of the catalog ID, user ID and prefix and may not be the same as the old (current) KDCFILE (the structure of the file name is described in section "The KDCFILE").

To ensure this, use the following procedure:

1. In the MAX statement, enter the file name without the *userid* in the *filebase* parameter. And under *prefix* (see "The KDCFILE") enter the same value as used for the generation of the “old” KDCFILE.
2. Start the KDCDEF run under a BS2000 user ID which is different to the one under which the application is running (for example, use userid2, if the old KDCFILE is called :catid:$userid1.prefix.KDCA).

You can subsequently - when the application is not running - copy the KDCFILE to the *userid1* and execute a KDCUPD run, if necessary. You can copy the KDCFILE using:

```
/COPY-FILE FROM-FILE=$userid2.filebase.KDCA,TO-FILE=$userid1.filebase.KDCA
```

Start the KDCDEF run under *userid1* or in MAX KDCFILE= enter the base name with the user ID,. This will cause KDCDEF to interrupt the KDCDEF run with the message K404 "DMS error D5B1 on file ...".

Unix, Linux and Windows systems:

You must ensure that the directory in which the new KDCFILE will be written is not the same directory as the base directory of the running UTM application.

To achieve this, proceed as follows:

1. Specify the base directory *filebase* with “.” in the MAX statement, i.e. the KDCFILE will be written in the directory in which KDCDEF is started:

   `MAX KDCFILE=(.,S)` or `MAX KDCFILE(.,D)`
2. You start KDCDEF in a directory other than the base directory of the UTM application.

You can then copy the KDCFILE to the base directory later and execute a KDCUPD run, if necessary.

If you start the KDCDEF run in the base directory or specify the fully qualified base directory in MAX KDCFILE= , then KDCDEF aborts the KDCDEF run with message U185.

2.4 Performance aspects - tuning

An important factor in the performance of a UTM application is the efficiency with which openUTM can access the KDCFILE, particularly in the case of high transaction rates. With a large configuration, i.e. a large KDCFILE, it is recommended that you optimize the access times. This can be achieved in two ways:

- splitting the KDCFILE (see below)
- KDCFILE on raw-device (only on Unix and Linux systems, see "KDCFILE on raw-device (Unix and Linux systems)")
- KDCFILE on stripe set (only on Windows systems, see "KDCFILE on a stripe set in (Windows systems)")

On BS2000 systems, you can also use the HIPERFILE concept to optimize performance, see the manual "BS2000 OSD/BC - Introductory Guide to DMS".

2.4.1 Splitting the KDCFILE

In order to improve the I/O behavior of your application, you can split the KDCFILE by swapping out the page pool and/or the restart area of the KDCFILE. Splitting the page pool and restart area across several files is particularly advantageous if you have very high transaction rates, since openUTM then distributes the number of access operations to these areas across the various different files.

The administrative data is essentially stored in the main file KDCA. The swapped-out page pool or restart area can be split between several files by means of generation. Provided this results in the use of numerous different hardware paths, access times can be reduced considerably thereby enhancing performance.

Generation notes

You can use the following operands of the KDCDEF control statement MAX to define the areas of the KDCFILE to be swapped out during generation, and to specify the number of files created for these areas:

- Page pool files

  ```
  MAX...,PGPOOLFS=number
  ```

- Restart area files

  ```
  MAX...,,RECBUFFS=number
  ```

In the case of dual-file operation, which is defined using the statement `MAX...,KDCFILE=(....,DOUBLE)`, these files are also maintained twice.

File names

The individual files of the KDCFILE that are contained in the swapped out areas have the same base name *filebase* as the main file KDCA and have the following names:

- Page pool files: P01A, P02A, P03A, ...
  If you are keeping a dual KDCFILE, the files P01B, P02B, P03B, ... are also created.
- Restart area: R01A, R02A, R03A, ...
  If you are keeping a dual KDCFILE, the files R01B, R02B, R03B, ... are also created.

Example

You want to set up your KDCFILE as follows:

- Page pool distributed across two files.
- Restart area located in a separate file.
- Duplicated file names.

For the base names, the place holder *FILEBASE* is used in this example.

On Unix, Linux and Windows systems, *FILEBASE* is the directory in which the files are stored, and can be replaced, for example, by `/home/userutm/base` (Unix and Linux systems) or `C:\userutm\base` (Windows systems).

In the KDCDEF generation you specify the following MAX statement:

```
MAX...,KDCFILE=(FILEBASE,DOUBLE),PGPOOLFS=2,RECBUFFS=1,...
```

KDCDEF then generates the following files:

	KDCFILE	Original	Copy
BS2000 systems:	Main file containing administrative data	*FILEBASE*.KDCA	*FILEBASE*.KDCB
	Page pool	*FILEBASE*.P01A *FILEBASE*.P02A	*FILEBASE*.P01B *FILEBASE*.P02B
	Restart area	*FILEBASE*.R01A	*FILEBASE*.R01B
Unix and Linux systems:	Main file containing administrative data	*FILEBASE*/KDCA	*FILEBASE*/KDCB
	Page pool	*FILEBASE*/P01A *FILEBASE*/P02A	*FILEBASE*/P01B *FILEBASE*/P02B
	Restart area	*FILEBASE*/R01A	*FILEBASE*/R01B
Windows systems:	Main file containing administrative data	*FILEBASE*\KDCA	*FILEBASE*\KDCB
	Page pool	*FILEBASE*\P01A *FILEBASE*\P02A	*FILEBASE*\P01B *FILEBASE*\P02B
	Restart area	*FILEBASE*\R01A	*FILEBASE*\R01B

2.4.2 KDCFILE on raw-device (Unix and Linux systems)

You can improve the performance of a UTM application on Unix and Linux systems considerably by operating the KDCFILE on raw-device, i.e. as a character based device file. To do this, create the KDCFILE on a separate disk partition, in other words a partition on which no file system is stored.

Direct access to the KDCFILE via a device file without buffering in the system kernel requires less time and less resources than access via the file system when the KDCFILE is stored as a normal file in the *filebase* directory. The KDCFILE is stored as a contiguous data area on the disk partition. If the KDCFILE is stored as a file within a file system, then the data of the KDCFILE is often stored by the system in such a way that it is distributed across several storage areas which leads to increased access times.

Splitting the KDCFILE across several files (swapping out the page pool and restart area) requires you to use a **separate** disk partition for each file.

> ¡ The raw partition of the database system used should be located on another disk.

Estimating the size of the required disk partition

To allow the system administrator to create a sufficiently large disk partition for your KDCFILE, you must calculate the size of the KDCFILE. This depends on the following factors:

- the number of generated objects addressed by name
  (transaction codes, users, program units, clients and printers, key sets, remote communication partners, connections for distributed processing, etc.)
- the generated size of the page pool (see "Page pool")
- the generated size of the restart area (see "Restart area")
- the number of work processes

To determine the size of the partition required for your KDCFILE, use KDCDEF to generate the KDCFILE as the file *filebase*/KDCA. Then output the size of your KDCFILE using the command *ls -l*. Please note that future modifications to the configuration generally affect the size of the KDCFILE. As a precaution, you should therefore select a larger disk partition.

Creating the raw special file

The disk partitioning is defined by the system administrator when installing the Unix or Linux system. Before system installation, therefore, you must inform your system administrator that you require disk partitions in raw-device format without file systems for your UTM applications. The system administrator can thus create several smaller partitions during installation, which can then be combined to form the storage area for your KDCFILE depending on requirements.

> ! **CAUTION!**
> Many disks contain administrative data in the first track. This area of the disk must therefore not be included in the partition for the KDCFILE.
>
> Do **not** create a file system in the disk partition in which the KDCFILE is to be stored. Do **not** mount the disk partition using the *mount* command.

Ask your system administrator to create a special file for accessing the disk partition. Make sure that access to the KDCFILE takes place via a character-oriented special file (raw-device), i.e. the name of the special file must begin with r and the identifier must be a c (first character output by the ls -l command).

The owner of the special file must the user ID under which the application runs. Read and write access to the special file must be granted exclusively to the owner (access rights 600). Be careful when assigning access rights to the KDCFILE, as these are the only means of protecting your KDCFILE against unauthorized access.

The *ls -l* command for the special file

```
ls -l /dev/rxxxx
```

returns the following output:

```
crw------- 1 utmaw other 0,1030 Jul 14 15:13 /dev/rxxxx
```

Writing the KDCFILE to the special file

There are two options for creating the KDCFILE in the disk partition.

- Delete the KDCFILE that you generated when determining the file size. Using the *ln* command, create a symbolic reference between the special file and *filebase*/KDCA. Regenerate the KDCFILE for your application using the KDCDEF generation tool. openUTM writes the KDCFILE directly to the special file.

  `rm` *filebase* `/KDCA`
  `ln -s /dev/rxxxx` *filebase* `/KDCA`
  utmpath `/ex/kdcdef`

- Copy the KDCFILE (generated when determining the file size) to the special file using the *cp* or *dd* command, and then delete the KDCFILE *filebase*/KDCA.
  Using the *ln* command, create a symbolic reference between the special file and *filebase*/KDCA.

  `cp` *filebase* `/KDCA /dev/rxxxx`
  `rm` *filebase* `/KDCA`
  `ln -s /dev/rxxxx` *filebase* `/KDCA`

In both cases, after issuing the *ln* command, use the *ls -l* command to check whether a link exists between *filebase* /KDCA and the special file:

ls -l /dev/rxxxx *filebase* `/KDCA`

`Output:`

```
crw-------  1 utmaw   other  0,1030  Jul  14  15:13  /dev/rxxxx
lrwxrwxrwx  1 utmaw   other  9       Jul  14  15:13  filebase /KDCA -> /dev/rxxxx
```

Dual-file operation

If you require dual-file operation of the KDCFILE for security reasons, you will need two disk partitions. The disk partitions should be located on different disks. Ideally, the disks should be operated by different controllers. The system administrator must create a raw special file for each KDCFILE.

To ensure that openUTM can write each KDCFILE to the special file created for this purpose, you must create the following symbolic references:

```
ln -s /dev/rxxx1 filebase /KDCA
ln -s /dev/rxxx2 filebase /KDCB
```

2.4.3 KDCFILE on a stripe set (Windows systems)

You create the *filebase* file directory on a stripe set (Windows software RAID level 0). In a stripe set, unused areas of matching size on different hard disks are combined to form a logical drive. The data in a KDCFILE on a stripe set are therefore also distributed amongst various hard disks, resulting in faster access and therefore in higher performance for the UTM application.

You must use stripe sets with parity (RAID Level 5) to achiever better data security. Stripe sets with parity can only be used under the Windows Server operating system.

Please consult the Windows documentation for more information on stripe sets.

3 Generating applications for distributed processing

This chapter provides a summary of the most important generation notes for applications with distributed processing and describes how the UTM generation is coordinated with the generation of the transport system.

The term distributed processing is used to describe server-server communication using the LU6.1 and OSI TP protocols. These protocols are used to implement global transaction processing. The OSI TP protocol also makes it possible to communicate with OpenCPIC clients and LU6.2 applications. OpenCPIC clients are generated in the same way as OSI TP partners, see chapter "Generation procedure for distributed processing based on OSI TP ". Specific hints for the generation of OpenCPIC clients can be found in section "Connecting OpenCPIC clients". More information about connecting to a LU6.2 application can be found in the openUTM manual “Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications”.

The basic principles of distributed processing are introduced in the openUTM manual “Concepts und Functions”.

To generate applications with distributed processing, you must first ensure that the individual applications have been generated without errors, and that the generation data of all applications involved has been coordinated. Since the KDCDEF generation tool can only check the generation data of a single application for consistency and syntactic accuracy, conflicts generally cannot be identified until the applications begin to interact with each other, e. g. during connection setup.

You will find notes on generation when standalone UTM applications are to be linked to UTM cluster applications on Unix, Linux or Windows systems in the sections "LU6.1-LPAP bundles of a standalone application with a UTM cluster application" and "OSI-LPAP bundles".

3.1 Distributed processing via the LU6.1 protocol

Before discussing the rules and recommendations for generating UTM applications with distributed processing, a number of SNA terms relevant for configuration are explained below in context.

SNA terms are shown in *italics* in the next section. More information on SNA terms can be found in the openUTM manual “Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications”.

3.1.1 Transport connections and SNA sessions

Communication between two applications is, from the openUTM point of view, carried out using transport connections (in the sense of TRANSDATA), via which the SNA sessions are handled.

The sessions are identified using *session names*. The session names serve to restart interrupted communication between two applications. If, prior to an interruption, communication via one of the possible transport connections is taking place using a given session name, then when the session is restarted, it is started under the same session name, but not necessarily using the same transport connection.

A session is defined using the KDCDEF control statement LSES, while its characteristics (e.g. the way in which it is opened, controlled, and managed) are defined using the SESCHA control statement.

The session name can be likened to the USER name in UTM applications: a USER can also continue an interrupted service on another terminal, thus using a different transport connection. In order to ensure that the session name in two connected applications does not have to be the same, the session name is made up of two parts (symbolized using the '+' character):

session-name = local-session-name+remote-session-name.

Each part of the session name is a maximum of 8 characters in length, thus the entire session name has maximum length of 16 bytes. The *local-session-name* refers to a common session in the local application, and the *remote-session-name* refers to the same session in the remote application. This means that the both the local and remote applications are required to know the session names of the partner application. The *local-session-name* uses the USER name defined in the local application to create a common name for the local application, and the *remote-sessionname* uses the USER names defined in the remote application to create a common name for the remote application.

At the start of a service, the “user ID" field in the KB header contains a local session name if the requester is a remote LU6.1 application.

A session is exclusively occupied for the duration of the dialog between the job-sending application and the job-receiving application (bracketing). In other words, another job-sending application will be required to:

- wait until the session has been released or
- repeat its job later, as it will be rejected at this time or
- occupy a different free session and start its job from there. The prerequisite for this is that there are several transport connections and sessions available for the remote application.

The opening and closing of a session is always controlled by one of the partner applications. This application is then referred to as the *primary logical unit* or *PLU*. However, the initiative for opening a session may come from **both** applications involved.

When opening a session, the partners agree on which application is to be responsible for for controlling the reservation of the session by jobs. The application which shall control the session is known as the contention winner, while the other application is referred to as the contention loser. In order to submit a job to the partner, the contention winner can reserve a session without consulting the contention loser beforehand. The contention loser, on the other hand, must request a session from the contention winner.

3.1.2 Generation notes

When generating UTM applications that are to communicate using the LU6.1 protocol, you must bear the following information in mind.

1. In each application, either one or two LPAP statements and the appropriate SESCHA, CON and LSES statements must be generated for each of the partner applications.

   Only one LPAP statement is required for a partner application if only one of the two applications is to be sending jobs. However, if both applications are intended to send jobs to the partner application, then both applications will require two LPAP statements.
2. An LPAP that is used mainly to send jobs is generated in its SESCHA statement using CONTWIN=NO; this ensures that the local application becomes the contention winner for this LPAP. The corresponding LPAP in the partner application must then be generated with CONTWIN=YES.
3. For each connection/each session, one CON or one LSES statement must be generated in each of the partner applications.

   For each CON statement, the CON name and the BCAMAPPL name in the one application must correspond to the names in the partner application.
   In the same way, for each LSES statement, the LSES name and the RSES name of the one application must correspond to those in the partner application.
4. In the case of standalone applications, the same number of CON and LSES statements must be generated for each LPAP; the number of CON or LSES statements determines the number of parallel connections to the partner application that are possible via this LPAP. Other rules apply to UTM cluster applications on Unix, Linux and Windows systems, see "Special issues with LU6.1 connections".
5. All CON and LSES statements of an LPAP must address the same partner application and must also be assigned to a single LPAP name in the partner application. It is not permitted to generate several CON statements leading to different applications for one LPAP name.

   It is also not permitted to generate several CON statements for a single LPAP which are assigned to different LPAP statements via the corresponding CON statements in the partner application.

   This generation error cannot be recognized by openUTM, but will lead to errors during connection or session establishment and during session restart.
6. In order to establish several parallel connections between two applications, a UTM application opens several transport system end points at the transport system. Each transport system end point of a UTM application is generated using its own BCAMAPPL statement.

   > **Unix, Linux and Windows systems**
   > Please note the maximum number of connections that can be established at a time via one transport system end point. For details see **BCAMAPPL statement** in section "BCAMAPPL - define additional application names".

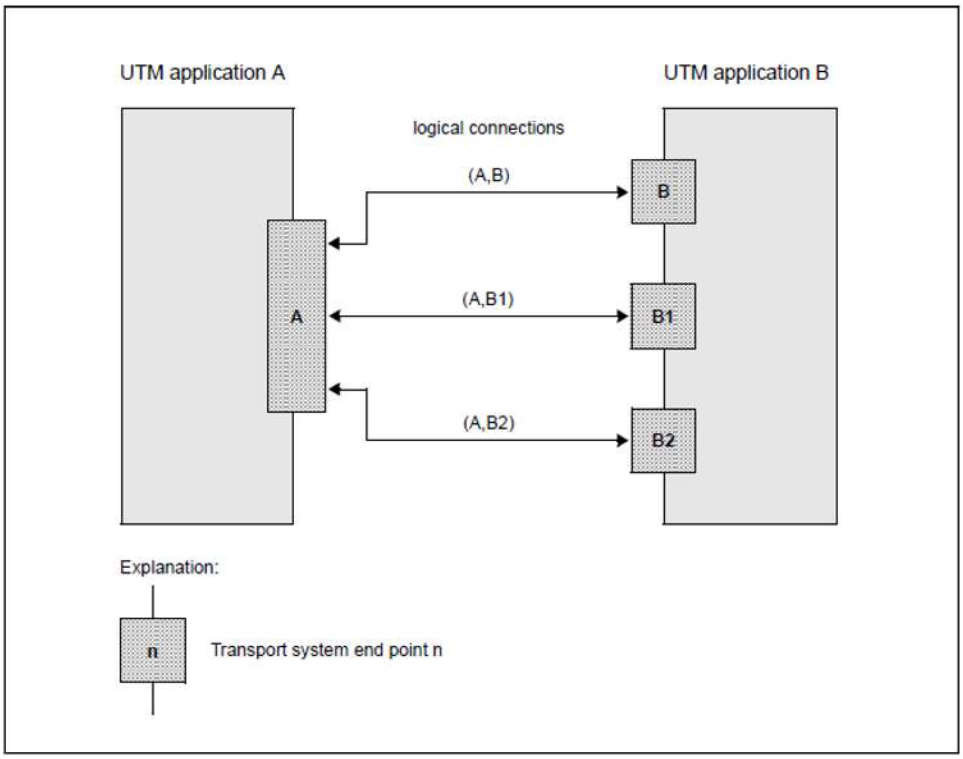


Figure 5: Two applications with several transport connections

In the example above, A and B are the names of the applications as specified using the MAX APPLINAME= statement; B1 and B2 are defined using separate BCAMAPPL statements.

Terminals are able to establish connections to application A using the application name A and to application B using the application name B. But application A is able to connect to application B using the application names B, B1 or B2.

BS2000 systems

> From the network administration point of view, the UTM application B consists of several BCAM applications. BCAM administration commands for one of the application names have an effect on the entire UTM application B. So in other words, a BCAPPL APPLICATION=B,MODE=DEACTIVATE command not only terminates UTM application B, but also signs the applications B1 and B2 off from BCAM.

Between two given transport system end points of both applications only a single transport connection can be established. If two transport system end points are generated in one of the applications and three in the other, then up to six parallel connections can be established between the two applications.

If both a contention winner LPAP and a contention loser LPAP are generated for a partner application (SESCHA statement), then transport system end points (BCAMAPPL statement) are established via the contention winner connections and should not be used simultaneously for the contention loser connections! This means that both contention winner and contention loser LPAPs are generated in a single application, thus the BCAMAPPLs of this application should be split into two disjunctive groups, where the BCAMAPPLs of the one group are assigned only to contention winner connections and the BCAMAPPLs of the other group are only ever used for contention user connections.

3.1.3 Procedure when generating LU6.1 connections

When generating two applications that are to communicate using the LU6.1 protocol, you should proceed as described below.

1. LPAP and SESCHA statements

   First you must decide whether the two applications are to be sending jobs to each other on an equally regular basis, or whether one of the applications is to be sending jobs more frequently than the other.

   In the first scenario, both applications must be generated with two LPAP statements each; in the second scenario the applications require one LPAP statement each. In this case, that LPAP statement that is designed to send more jobs than it receives is generated with SESCHA ...,CONTWIN=NO; the corresponding LPAP in the partner application is generated with SESCHA ...,CONTWIN=YES. When you have two LPAP statements in an application, one should be generated with SESCHA ... ,CONTWIN=NO and the other with SESCHA ..., CONTWIN=YES.

   LPAP statement in section "LPAP - define an LPAP partner for distributed processing based on LU6.1"

   The following operands can be used to define an LPAP partner as the logical connection point for the partner application.

   - *lpapname*

     LPAP partner name; this is the logical name of the partner application via which the program units of the local application and the partner application communicate. *lpapname* is only significant in the local application.

   - SESCHA=

     The session characteristics for communication between local application and partner application as defined under *sescha_name* in the SESCHA statement are assigned to the LPAP partner.

   - PERMIT=

     Specifies the level of authorization (right to carry out administration and preselection functions) of the partner application.

   - QLEV=

     Specifies the maximum number of asynchronous messages that are permitted to wait in the Message Queue of the LPAP partner.

   - DEAD-LETTER-Q=

     Specifies whether asynchronous messages to the LPAP partner that are deleted as they could not be sent due to a permanent error are saved in the dead letter queue.

   - STATUS=

     This defines whether the partner application is able to work with the local application immediately the local application is started, or whether the administrator must first set the status to ON.

   - BUNDLE=

     Makes the LPAP a slave LPAP of an LU6.1-LPAP bundle and specifies the associated master LPAP.

SESCHA statement in section "SESCHA - define session characteristics for distributed processing based on LU6.1"

You can use the following operands to define the session characteristics that are to be assigned to one of the LPAP partners and therefore to the partner application that connects via this LPAP partner.

- *sescha_name*

  Defines the name under which the session characteristics are collected. This name is entered in the LPAP statement in the operand SESCHA= to assign the session characteristics to a LPAP partner.

- CONTWIN=

  Specifies whether the local application is the contention winner (NO) or contention loser (YES). The contention winner application manages the session and controls the utilization of the session by jobs.

  Default: If PLU=N, the local application is the contention loser, otherwise it is the contention winner.

- PLU=

  Specifies which application is able to initiate the session, or in other words, whether the partner application is the primary logical unit (PLU) (YES) or the local application (NO).
  PLU=Y must be specified for one of the participating applications, and PLU=N for the other.

- CONNECT=

  Specifies whether the local application is to connect automatically to the partner application on application startup (YES) or whether the connection to the partner application is to be carried out by means of an administration command (NO).

Example

Application A sends jobs to Application B via LPAP B1 and application B sends jobs to application A via LPAP A2.

Application A:	Application B:

`LPAP B1, SESCHA=B1`
`SESCHA B1, CONTWIN=NO, PLU=YES`
 	
`LPAP B2, SESCHA=B2`
`SESCHA B2, CONTWIN=YES, PLU=NO`
	`LPAP A1, SESCHA=A1`
	`SESCHA A1, CONTWIN=YES, PLU=NO`
	
	`LPAP A2, SESCHA=A2`
	`SESCHA A2, CONTWIN=NO, PLU=YES`

2. BCAMAPPL statements

   The next thing to do is to specify how many parallel connections are to be generated between two LPAPs. In accordance with the number you specify, the BCAMAPPL statement is used to generate additional transport system endpoints for both applications. Only one connection may be established between each transport system endpoint and the transport system endpoint of the partner application. Should, for example, nine parallel connections be generated between two LPAPs, then at least three BCAMAPPL statements will be required on each side.

   If two LPAPs to the partner application are generated in an application, the BCAMAPPLs of this application should be split into two disjunct groups. The first LPAP communicates using just the BCAMAPPLs of the first group, and the second LPAP uses the BCAMAPPLs of the second group.

BCAMAPPL statement in section "BCAMAPPL - define additional application names"
The following operands are used to define an additional application name for parallel connections to the communication partner.

- *appliname*

  Additional (BCAM) name of the UTM application.

- T-PROT
  Specifies the transport protocol.
  NEA is the default for BS2000, RFC1006 is the default for Unix, Linux and Windows systems.

If an application communicates with several partner applications, the BCAMAPPLs used to communicate with the one application may also be used to communicate with the other applications.

3. CON and LSES statements

   It is then necessary to assign one CON and one LSES statement to each LPAP statement for each parallel connection via this LPAP. Each CON and each LSES statement must be generated in each of the participating applications and both of these generations must correspond to each other.

   Thus:

   - Each CON name in the one application corresponds to a BCAMAPPL name in the other application and vice versa.
   - Each LSES name in the one application corresponds to an RSES name in the other application and vice versa.

CON statement in section "CON - define a connection for distributed processing based on LU6.1"
The following operands can be used to assign the LPAP partner in the local application to the real partner application.

- *remote_appliname*

  Name of the partner application with which communication is to take place via the logical connection.

- BCAMAPPL=

  Refers to a name of the local application as specified in the control statement MAX or BCAMAPPL. You cannot specify a BCAMAPPL name for which a T-PROT=SOCKET has been generated.

- LPAP=

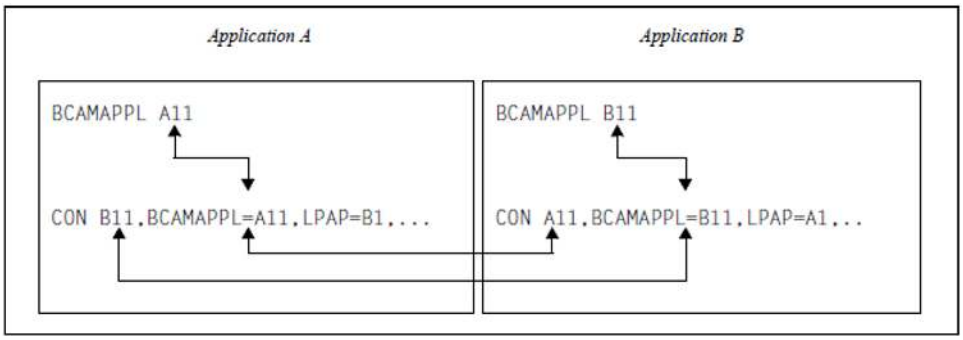
  Name of the LPAP partner application to which the connection is to be established. The name of the LPAP partner via which the partner application connects must be defined using the statement LPAP *lpapname*.
  Specifying several CON statements with the same *lpapname* allows you to generate parallel connections to the partner application.

- PRONAM=

  Name of the partner computer.

The CON statements that are used to describe the connection to the partner application in the local application and the connection to the local application in the partner application refer to the **same** connection. CON statements must therefore always be entered in pairs. When using parallel sessions, several CON statements are generated for an LPAP partner.

In the example, the assignment is made between the LPAP partner B1 (as generated in application A) and the LPAP partner A1 (as generated in application B):



LSES statement in section "LSES - define a session name for distributed processing based on LU6.1"

The following operands can be used to agree the same session names for the connection and assign them to the LPAP partner.

- *local_sessionname*

  Name of the session in the local application.

- RSES=

  Name of the session in the partner application.

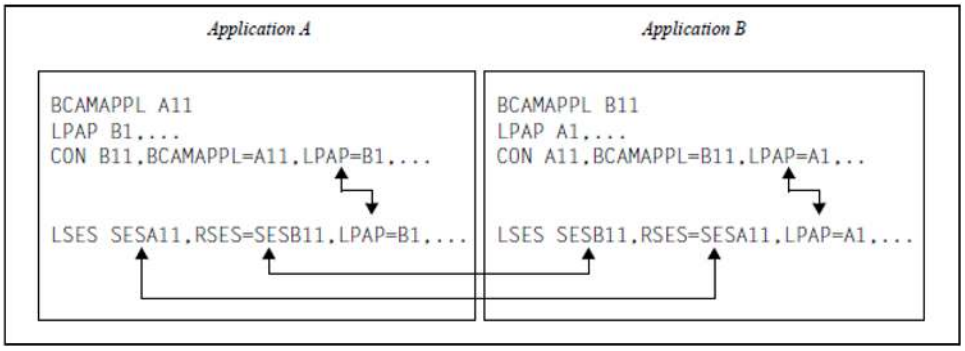
- LPAP=

  Name of the LPAP partner that is assigned to the partner application.
  local_sessionname is used for communication with the partner application that is assigned to the LPAP partner *lpapname* in the local application.

Session names are agreed in the local application and partner application. LSES statements must therefore always be entered in pairs. Since the session name is assigned to the LPAP partners, the LPAP partner assignment defined in the LSES statement must be identical to that defined in the CON statements.

If two LPAP partners are assigned to each other, the LSES and RSES names agreed in the LSES statements must match (see example below). In the case of parallel sessions, several LSES statements are entered with different session names for an LPAP partner *lpapname*.

The previous example can now be extended as follows.



Example

Application A:	Application B:
BCAMAPPL A11 BCAMAPPL A12 LPAP B1, SESCHA=B1 SESCHA B1, CONTWIN=NO, PLU=YES	BCAMAPPL B11 BCAMAPPL B12 LPAP A1, SESCHA=A1 SESCHA A1, CONTWIN=YES, PLU=NO
CON B11, BCAMAPPL=A11, LPAP=B1 CON B12, BCAMAPPL=A11, LPAP=B1 CON B11, BCAMAPPL=A12, LPAP=B1 CON B12, BCAMAPPL=A12, LPAP=B1 LSES SESA11, RSES=SESB11, LPAP=B1 LSES SESA12, RSES=SESB12, LPAP=B1 LSES SESA13, RSES=SESB13, LPAP=B1 LSES SESA14, RSES=SESB14, LPAP=B1	CON A11, BCAMAPPL=B11, LPAP=A1 CON A11, BCAMAPPL=B12, LPAP=A1 CON A12, BCAMAPPL=B11, LPAP=A1 CON A12, BCAMAPPL=B12, LPAP=A1 LSES SESB11, RSES=SESA11, LPAP=A1 LSES SESB12, RSES=SESA12, LPAP=A1 LSES SESB13, RSES=SESA13, LPAP=A1 LSES SESB14, RSES=SESA14, LPAP=A1
BCAMAPPL A21 BCAMAPPL A22 LPAP B2, SESCHA=B2 SESCHA B2, CONTWIN=YES, PLU=NO	BCAMAPPL B21 BCAMAPPL B22 LPAP A2, SESCHA=A2 SESCHA A2, CONTWIN=NO, PLU=YES
CON B21, BCAMAPPL=A21, LPAP=B2 CON B22, BCAMAPPL=A21, LPAP=B2 CON B21, BCAMAPPL=A22, LPAP=B2 CON B22, BCAMAPPL=A22, LPAP=B2 LSES SESA21, RSES=SESB21, LPAP=B2 LSES SESA22, RSES=SESB22, LPAP=B2 LSES SESA23, RSES=SESB23, LPAP=B2 LSES SESA24, RSES=SESB24, LPAP=B2	CON A21, BCAMAPPL=B21, LPAP=A2 CON A21, BCAMAPPL=B22, LPAP=A2 CON A22, BCAMAPPL=B21, LPAP=A2 CON A22, BCAMAPPL=B22, LPAP=A2 LSES SESB21, RSES=SESA21, LPAP=A2 LSES SESB22, RSES=SESA22, LPAP=A2 LSES SESB23, RSES=SESA23, LPAP=A2 LSES SESB24, RSES=SESA24, LPAP=A2

Notes

- In the case of applications with distributed processing, the *length* value specified in the statement MAX..., RECBUF=(*number*,*length*),... may have to be increased. Further information can be found in section "Restart area".
- The behavior of an application can be influenced by the choice of timer (operand IDLETIME= of the SESCHA statement, operands CONCTIME and PTCTIME of the UTMD statement).

3.1.4 LU6.1-LPAP bundles

LU6.1=LPAP bundles allow messages to be distributed automatically across several LPAP partners. 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 LU6.1-LPAP bundle, openUTM is responsible for distributing the messages to the partner application instances. To achieve this, the program units in the APRO call must address the MASTER-LU61-LPAP.

One application scenario for distributing messages in this way is communication between a UTM application and a UTM cluster application. This allows messages to the UTM cluster application to be distributed across the individual node applications. You will find detailed information on this in section "LU6.1-LPAP bundles of a standalone application with a UTM cluster application".

An LU6.1-LPAP bundle consists of a master LPAP and multiple slave LPAPs. The slave LPAPs are assigned to the master LPAP on generation. In normal circumstances, the individual slave LPAPs address different partner applications.

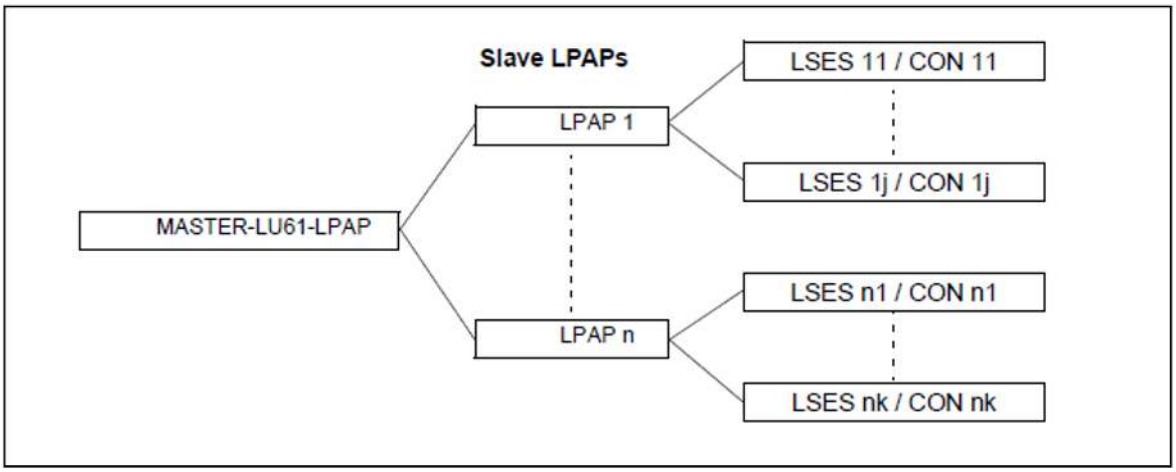


Figure 6: Example of an LU6.1-LPAP bundle

Generating an LU6.1-LPAP bundle

MASTER-LU61-LPAP statement in section "MASTER-LU61-LPAP - Define the master LPAP of an LU6.1-LPAP bundle"
Specifies the name and properties of the master LPAP in an LU6.1-LPAP bundle.

- *master-lpap-name*

  Name of the master LPAP.
- STATUS=

  Specifies whether messages can be sent to this LPAP bundle.

LPAP statement in section "LPAP - define an LPAP partner for distributed processing based on LU6.1"
The following properties must be specified to generate a slave LPAP:

- *lpap-name*

  Name of the slave LPAP.

- BUNDLE=master-lpap-name

  Name of the master LPAP. The master LPAP specified here must be defined in a MASTER-LU61-LPAP statement. If you specify BUNDLE, this LPAP becomes a slave LPAP of the specified master LPAP.

  ```
  MASTER-LU61-LPAP master , ...
  LPAP slave-lpap , BUNDLE= master , ...
  ```

CONs of LPAPs belonging to an LU6.1-LPAP bundle

- No physical connections (CONs) can be assigned to a master LPAP. This means that it cannot be specified as the LPAP in a CON statement. The master LPAP always uses the connections assigned to the slave LPAPs.

Distribution of messages

For details, refer to the section "Distributing messages" in chapter "LU6.1-LPAP bundles".

Display in the KB header

For details, refer to the section "Information displayed in the KB header" in chapter "LU6.1-LPAP bundles".

3.1.5 Usage of LU6.1-LPAP bundles for communication with an UTM cluster application on Unix, Linux and Windows systems

Note the following when generating LU6.1 communication between a standalone partner application and a UTM cluster application:

- A partner application must generate one LPAP with a specific number of sessions and connections for each node of the UTM cluster application with which it wants to communicate.
- To address the UTM cluster application, an LU6.1-LPAP bundle whose slave LPAPs are assigned to the cluster node should be generated in the partner application (see "MASTER-LU61-LPAP – Define the master LPAP of an LU6.1-LPAP bundle").
- In the UTM cluster application, more sessions (LSES) than connections (CON) must be generated for the LPAP that represents the partner application: One session per cluster node must be generated for each connection.

  Each cluster node requires only exactly the number of connections assigned to each LPAP in the partner application for the corresponding LPAP. However, because all cluster nodes have identical generations, the sessions for all the LPAPs of the partner application must be generated in every cluster node.
- During generation, the LU6.1 sessions must be explicitly assigned to the node applications. To do this, define the reference name of the node application in the NODE-NAME parameter of the CLUSTER-NODE statement and specify this name in the NODE-NAME parameter of the LSES statement. As a result, the "right" session is selected when a session is established with a partner application.

Example:

> The example below shows a generation in which the standalone application SA on the host HOSTSA is linked to the UTM cluster application CA on the cluster nodes NODECAX, NODECAY and NODECAZ. 4 connections between the standalone application and each of the node applications are generated. A MASTER-LU61-LPAP is generated for the LPAPs that represent the node applications in the standalone application. This represents the UTM cluster application.

Standalone application SA on HOSTSA	UTM cluster application CA on NODECAX/Y/Z
	CLUSTER-NODE NODE-NAME=NODECAX - , HOSTNAME=NODECAX, - , FILEBASE=BASE1 CLUSTER-NODE NODE-NAME=NODECAY - , HOSTNAME=NODECAY, - , FILEBASE=BASE2 CLUSTER-NODE NODE-NAME=NODECAZ - , HOSTNAME=NODECAZ, - , FILEBASE=BASE3
BCAMAPPL SA11 BCAMAPPL SA12 MASTER-LU61-LPAP MLPAPCA LPAP LPAPCAX,SESCHA=SESCHCA- , BUNDLE=MLPAPCA LPAP LPAPCAY,SESCHA=SESCHCA- , BUNDLE=MLPAPCA LPAP LPAPCAZ,SESCHA=SESCHCA- , BUNDLE=MLPAPCA SESCHA SESCHCA, CONTWIN=NO, PLU=YES	BCAMAPPL CA11 BCAMAPPL CA12 LPAP LPAPSA, SESCHA=SESCHSA SESCHA SESCHSA, CONTWIN=YES, PLU=NO
CON CA11, PRONAM=NODECAX – , BCAMAPPL=SA11, LPAP=LPAPCAX CON CA12, PRONAM=NODECAX – , BCAMAPPL=SA11, LPAP=LPAPCAX CON CA11, PRONAM=NODECAX – , BCAMAPPL=SA12, LPAP=LPAPCAX CON CA12, PRONAM=NODECAX – , BCAMAPPL=SA12, LPAP=LPAPCAX	CON SA11, PRONAM=HOSTSA – , BCAMAPPL=CA11, LPAP=LPAPSA CON SA11, PRONAM=HOSTSA – , BCAMAPPL=CA12, LPAP=LPAPSA CON SA12, PRONAM=HOSTSA – , BCAMAPPL=CA11, LPAP=LPAPSA CON SA12, PRONAM=HOSTSA – , BCAMAPPL=CA12, LPAP=LPAPSA
CON CA11, PRONAM=NODECAY – , BCAMAPPL=SA11, LPAP=LPAPCAY CON CA12, PRONAM=NODECAY – , BCAMAPPL=SA11, LPAP=LPAPCAY CON CA11, PRONAM=NODECAY – , BCAMAPPL=SA12, LPAP=LPAPCAY CON CA12, PRONAM=NODECAY – , BCAMAPPL=SA12, LPAP=LPAPCAY	

Standalone application SA on HOSTSA	UTM cluster application CA on NODECAX/Y/Z
CON CA11, PRONAM=NODECAZ – , BCAMAPPL=SA11, LPAP=LPAPCAZ CON CA12, PRONAM=NODECAZ – , BCAMAPPL=SA11, LPAP=LPAPCAZ CON CA11, PRONAM=NODECAZ – , BCAMAPPL=SA12, LPAP=LPAPCAZ CON CA12, PRONAM=NODECAZ – , BCAMAPPL=SA12, LPAP=LPAPCAZ	
LSES SAA2CAX, RSES= CA12SA1- , LPAP=LPAPCAX LSES SAB2CAX, RSES= CA12SA2- , LPAP=LPAPCAX LSES SAC2CAX, RSES= CA12SA3- , LPAP=LPAPCAX LSES SAD2CAX, RSES= CA12SA4- , LPAP=LPAPCAX	LSES CA12SA1, RSES= SAA2CAX- , LPAP=LPAPSA - , NODE-NAME=NODECAX LSES CA12SA2, RSES= SAB2CAX- , LPAP=LPAPSA - , NODE-NAME=NODECAX LSES CA12SA3, RSES= SAC2CAX- , LPAP=LPAPSA , NODE-NAME=NODECAX LSES CA12SA4, RSES= SAD2CAX- , LPAP=LPAPSA - , NODE-NAME=NODECAX
LSES SAA2CAY, RSES= CA22SA1- , LPAP=LPAPCAY LSES SAB2CAY, RSES= CA22SA2- , LPAP=LPAPCAY LSES SAC2CAY, RSES= CA22SA3- , LPAP=LPAPCAY LSES SAD2CAY, RSES= CA22SA4- , LPAP=LPAPCAY	LSES CA22SA1, RSES= SAA2CAY- , LPAP=LPAPS - , NODE-NAME=NODECAY LSES CA22SA2, RSES= SAB2CAY- , LPAP=LPAPSA - , NODE-NAME=NODECAY LSES CA22SA3, RSES= SAC2CAY- , LPAP=LPAPSA - , NODE-NAME=NODECAY LSES CA22SA4, RSES= SAD2CAY- , LPAP=LPAPSA - , NODE-NAME=NODECAY

Standalone application SA on HOSTSA	UTM cluster application CA on NODECAX/Y/Z
LSES SAA2CAZ, RSES= CA32SA1- , LPAP=LPAPCAZ LSES SAB2CAZ, RSES= CA32SA2- , LPAP=LPAPCAZ LSES SAC2CAZ, RSES= CA32SA3- , LPAP=LPAPCAZ LSES SAD2CAZ, RSES= CA32SA4- , LPAP=LPAPCAZ	LSES CA32SA1, RSES= SAA2CAZ- , LPAP=LPAPSA - , NODE-NAME=NODECAZ LSES CA32SA2, RSES= SAB2CAZ- , LPAP=LPAPSA - , NODE-NAME=NODECAZ LSES CA32SA3, RSES= SAC2CAZ- , LPAP=LPAPSA - , NODE-NAME=NODECAZ LSES CA32SA4, RSES= SAD2CAZ- , LPAP=LPAPSA - , NODE-NAME=NODECAZ

3.2 Distributed processing via the OSI TP protocol

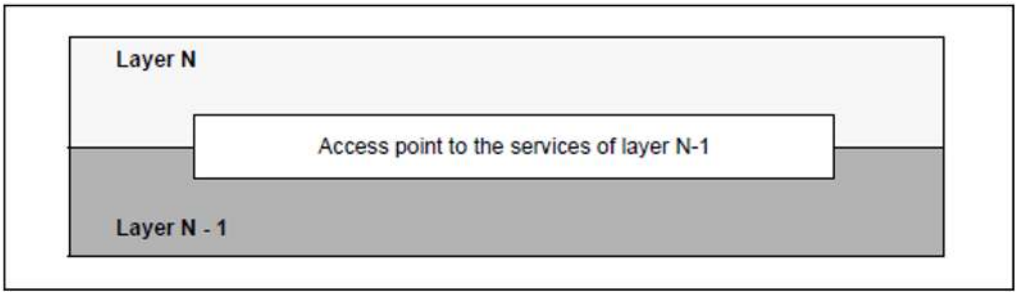
Before discussing the rules and recommendations for generating UTM applications with distributed processing via OSI TP, a number of OSI terms relevant for configuration are explained below. The OSI terms in this section are shown in *italics*.

3.2.1 OSI terms

If two partners wish to communicate with each other, the rules they must observe and the services they must provide have been standardized in the OSI protocol (Open System Interconnection).

ISO (International Organization for Standardization) has also developed the *OSI reference model*, in which the various communication tasks are distributed over *seven layers (instances)*. The services to be provided by each layer are clearly defined. The seven layers form a hierarchical structure, where each layer can access the *services* of the underlying layer. These services are made available to the overlying layer at *service access points*.

During communication, the application accesses the services of the communication system via one of these access points:



If two applications wish to communicate and exchange data, they must be linked via a *transport connection*. A transport connection can only be established between two *addressable units* in the network. It must therefore be possible to identify each application by means of an *address* which is unique throughout the network.

In the OSI world, addresses are assigned to service access points rather than applications. Within the network, the application is thus identified by means of the address of the access point via which it communicates.

Each service access point is assigned an address which is unique throughout the network. This consists of a *selector* and the address of the underlying access point.

The following diagram shows the format of addresses in the OSI reference model.

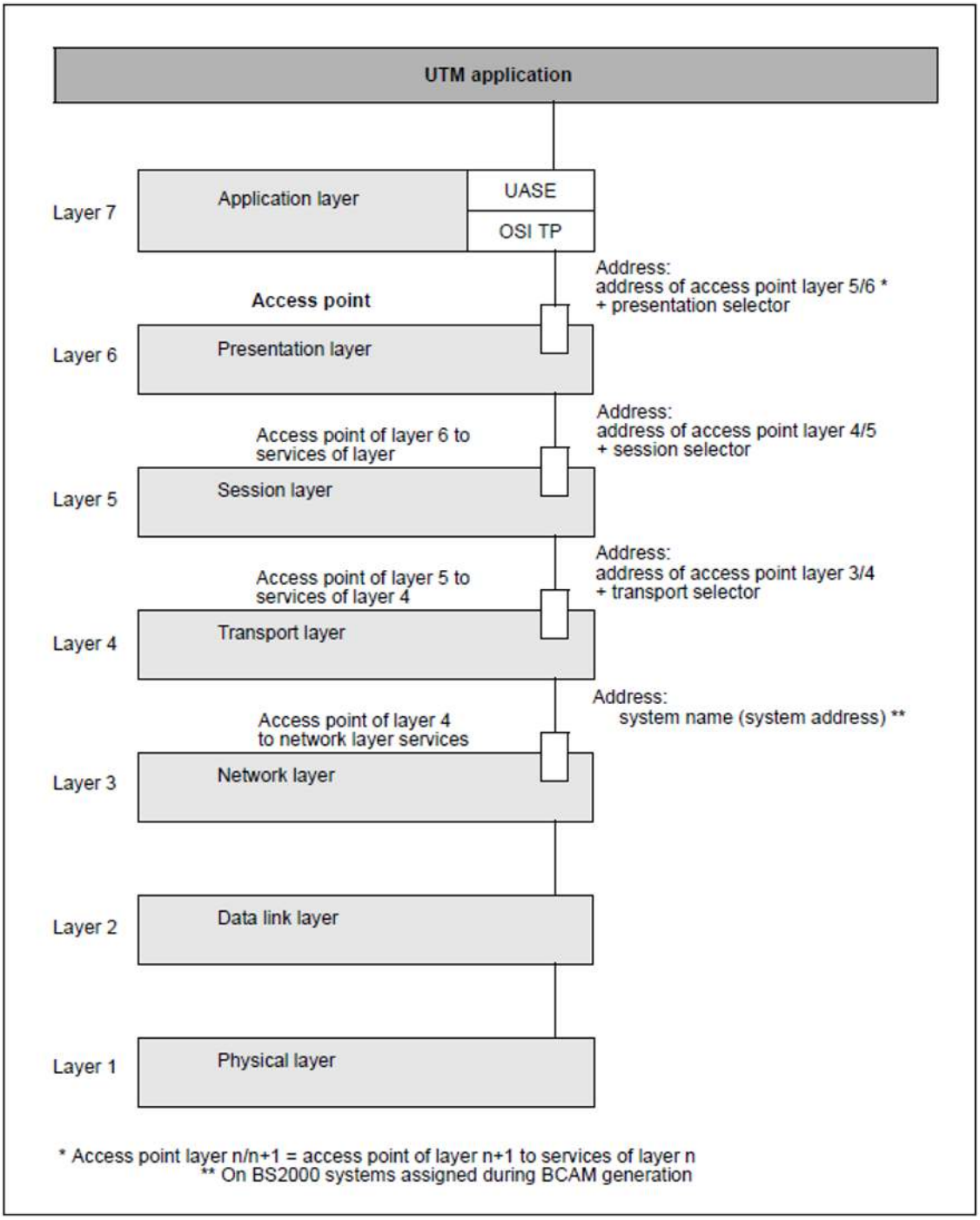


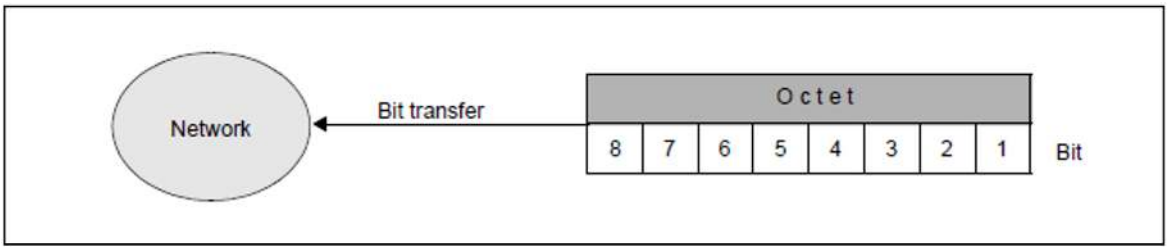
Figure 7: Addresses of the service access points in the OSI reference model

In order to communicate with other applications in the network, a UTM application links to a service access point. This link is generated using the ACCESS-POINT statement. The UTM application can then be accessed by its partners in the network via the address of this access point.

The format of the access point address via which the application is accessed depends on the access point hierarchy. If the UTM application communicates on the basis of OSI TP, it links to an access point to the services of layer 6. The address of the access point in the local system consists of the *transport selector* (selector of the *transport layer*), the *session selector* (selector of the *session layer*), and the *presentation selector* (selector of the *presentation layer*). The network address of the access point thus comprises the network address of the system and the address of the access point in the local system.

The selectors of the individual layers must be unique in the local system. The system address is unique throughout the entire network. When defining the access point address, you must consult your network administrator.

The selectors consist of *octets*. An octet is a byte (8 bits) in which the bit numbering and thus the order in which bits are transferred is fixed.



Order of bit transfer in an octet

It is possible to establish several *parallel connections* (also known as *associations*) to a certain communication partner. All connections to a remote partner are generated in a **single** OSI-CON statement. The OSI-LPAP statement can be used to define the number of parallel connections to a particular partner. Each individual connection must be assigned an *association name*, which is unique throughout the local system. The names of associations with a remote partner are also generated in the OSI-LPAP statement.

Each connection between two partners is managed by one of the partners. This partner is known as the *contention winner*, while the other partner is referred to as the *contention loser*. Jobs can be initiated by both partners. If both partners submit a job at the same time, priority is given to the contention winner. The contention winner of a connection should be the communication partner that starts jobs most frequently.

If several parallel connections exist between two partner, it is not necessary to define which partner is the contention loser and which partner is the contention winner for each connection. You simply specify the number of connections for which the individual partners act as the contention winner (OSI-LPAP statement). The partner that starts jobs most frequently should be defined as the contention winner.

openUTM supports *TPSU-title* (**T**ransaction **P**rocessing **S**ervice **U**ser). This is an OSI TP user which provides certain services within an application. In openUTM this is the sequence of program units which form a service. A TPSU-title is a unique name within the application. In openUTM it is the TAC name of the first program unit of a service. The *initiating TPSU-title* is the TPSU-title of the job submitter and the *recipient TPSU-title* is the TPSU-title of the job receiver.

openUTM supports the *application entity title* (AET) defined by ISO. This is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET in order to establish a connection. In openUTM, the AET is specified for information purposes, but it is not used for addressing the partner.

The *application context* to be used for communication purposes must be defined for each remote partner with which the UTM application wishes to communicate via the OSI TP protocol.

The OSI terms *application entity title* and *application context* are described in further detail in the following sections.

Application entity title (AET)

In the OSI world, communication partners are represented by application entities. An application entity is an addressable unit in layer 7 of the OSI reference model (application layer). An example would be the access point of a UTM application, via which an OSI TP communication partner can link to the UTM application. In the OSI TP standard, each application entity is assigned an application entity title, which can be used to uniquely address the application entity in the OSI network.

The ISO standard defines two forms of AET: the directory form and the object identifier form. The latter is supported by openUTM. This is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET in order to establish a connection. In the case of homogeneous communication between UTM and UTM, the AET is also specified, but is not used for addressing the partner. It consists of two parts:

- the application process title (APT)
- the application entity qualifier (AEQ)

Application process title (APT)

The APT is used to identify the application. In accordance with the ISO standard, it must be unique globally (i.e. worldwide). For this reason, it should be assigned and registered by a standardization body, e.g. in Germany this is the Deutsche Gesellschaft for Warenkennzeichnung GmbH (DGWK = Germany company for registering trademarks).

An APT in object identifier form consists of up to 10 components:

`(component1,component2,...,component10)`

Some of the values for *component1* through *component10* have been standardized. Here, symbolic names have been assigned to certain numbers. The value range for *component2* depends on the value for *component1*. The table below lists the symbolic names and value ranges supported by openUTM:

component1	`0:CCITT`	`1:ISO`	`2:JOINT-ISO-CCITT`

component2	`0:RECOMMENDATION`
	`1:QUESTION`
	`2:ADMINISTRATION`
	`3:NETWORK-OPERATOR`
		`0:STANDARD`
		`1:REGISTRATION-AUTHORITY`
		`2:MEMBER-BODY`
		`3:IDENTIFIED-ORGANIZATION`	

	Value range: 0 - 39	Value range: 0 - 39	Value range: 0 - 67 108 863
component3 through component10	Value range: 0 - 67 108 863	Value range: 0 - 67 108 863	Value range: 0 - 67 108 863

The APT specified in openUTM need not be assigned by a standardization body, i.e. it is freely selectable. However, it must meet the following two requirements:

- it must be unique within the network
- it must contain permitted values, as shown in the table above

Application entity qualifier (AEQ)

The AEQ identifies an access point within an application. You can only assign AEQs to the access points of an application if you have assigned an APT to the application itself.

The AEQ is a positive integer between 0 and 67108863.

The AEQ must be unique within the application, i.e. the application must not contain two access points with the same AEQ. However, it is not necessary to assign an AEQ to all access points in the application.

When there are parallel associations and a connection is being established, the AEQ is checked to see if it is the same one as used for the first association established.

Application context

The application context to be used for communication purposes must be coordinated with each partner application with which your local application wishes to communicate via the OSI TP protocol.

The application context must be explicitly defined for each partner application. It determines the rules governing the transfer of messages between the local application and partner application. openUTM supports the following predefined application contexts:

- UDTAC
- UDTDISAC
- XATMIAC
- UDTCCR
- UDTSEC
- XATMICCR

If you are not using one of the application contexts listed above you can use the APPLICATION-CONTEXT statement which is described the section "APPLICATION-CONTEXT - define the application context" to generate further application contexts.

All the involved partners must agree the following when using an application context:

- An *abstract syntax*, which defines how the user data is encoded for transfer. By default, openUTM supports the following abstract syntaxes:
  - UDT (Unstructured Data Transfer)
  - XATMI
  - CCR
  - UTMSEC

  See also the ABSTRACT-SYNTAX statement in section "ABSTRACT-SYNTAX - define the abstract syntax".
- A *transfer syntax*, which defines the format in which the user data is transferred. By default, openUTM supports the transfer syntax Basic Encoding Rules (BER).

  See also the TRANSFER-SYNTAX statement in section "TRANSFER-SYNTAX - define the transfer syntax".

Both communication partners must generate the same abstract syntaxes as the application context used for communication. If the application context generated locally is not identical to that generated in the partner, openUTM rejects any attempts to establish the association with a corresponding message.

You only need to use the ABSTRACT-SYNTAX, TRANSFER-SYNTAX and APPLICATION-CONTEXT statements if you are not using any of the standard application contexts made available by openUTM.

3.2.2 Generation of distributed processing based on OSI TP

The following KDCDEF statements are provided for generating the communication partners of an application and the connections to these partners:

Statement	Function
ABSTRACT-SYNTAX	Define the abstract syntax for the user data: • assign a unique object identifier • assign the transfer syntax for data transfer
ACCESS-POINT	Define the name and address of the local OSI TP access point: • define the application entity qualifier (AEQ) of the local application (address component of the application entity title)
APPLICATION-CONTEXT	Define the application context for communication with the partner application: • assign the abstract syntax for the user data • assign a unique object identifier
LTAC	Assign local TAC names for services in the partner application, under which these services are then started locally
MASTER-OSI-LPAP	Define the name and properties of the master LPAP in a OSI-LPAP bundle (see "OSI-LPAP bundles")
OSI-CON	Define connections between the local application and the remote partner and assign these to the OSI-LPAP partner: • specify a local OSI TP access point • specify the network address of the partner application

Statement	Function
OSI-LPAP	Define an OSI-LPAP partner as the logical access point for the partner application: • specify the application entity title (i.e. APT and AEQ) of the partner application • specify the application context of the partner application • define the number of (parallel) connections to the partner and the names of these connections • define the number of connections to be established automatically when the application is started • define the number of connections for which the local application is to act as the contention winner • define the access rights of the partner application in the local application • define the administration authorization level of the partner application • define maximum values for the message queue of the OSI-LPAP partner • define the status of the OSI-LPAP partner on connection setup • if necessary, make the OSI-LPAP a slave LPAP of a OSI-LPAP bundle and specify the associated master LPAP • If necessary, specify whether asynchronous messages to the OSI-LPAP partner that are deleted as they could not be sent due to a permanent error are saved in the dead letter queue.
TRANSFER-SYNTAX	Define the transfer syntax for data transfer: • assign a unique object identifier
UTMD	Define global values and the address of the local UTM application: • define the application process title (APT) (address component of the application entity title) • define the maximum waiting time for establishing an association • define the maximum waiting time for confirmation of asynchronous messages
The following additional parameters are available on Unix, Linux and Windows systems:	
MAX XAPTPSHMKEY	Define an authorization key for the XAPTP shared memory segment.
MAX OSISHMKEY	Define an authorization key for the OSS shared memory segment
MAX OSI-SCRATCH-AREA	Define the size of the working area for dynamic data storage

To allow for communication based on the OSI TP protocol, you must perform the following steps:

- Generate the application entity title (AET)

  The statement UTMD...,APPLICATION-PROCESS-TITLE= is used to define the application process title (APT) as the address component of the AET for your application. A remote partner that requires AETs must know this APT in order to establish a connection.

  The application entity title is assigned to the OSI-LPAP partner. In the OSI-LPAP statement, use the APPLICATION-PROCESS-TITLE= operand to specify the APT and the APPLICATION-ENTITY-QUALIFIER= operand to specify the AEQ of the access point for the partner application. The AEQ must already be generated for the access point in the remote partner application.

  Using the statement ACCESS-POINT...,APPLICATION-ENTITY-QUALIFIER=, define the application entity qualifier (AEQ) as the address component of the AET for the access point of the local application. A partner application must know the AEQ of the access point via which communication takes place with the local application.

  > **i** The following applies to UTM cluster applications:If you specify an APT with less than 10 elements in the UTMD statement for an OSI-TP link then UTM adds an index (1, 2 etc.) to the APT generated for each node application. This guarantees that the AET is unique. You are therefore recommended to generate no more than 9 elements in UTM cluster applications.

- Define the application context for communication with the partner application

  If you do not wish to work with one of the default application contexts listed in section "OSI terms", you can generate the application context to be used for communication with the partner application using the APPLICATION-CONTEXT statement. This involves assigning defined abstract syntaxes and a unique object identifier to the application context.

  The ABSTRACT-SYNTAX statement serves to specify the abstract syntax used for the transfer of user data and to assign a unique object identifier to this abstract syntax.

  The transfer syntax (which defines how the user data is to be encoded and decoded for data transfer) is also defined using the ABSTRACT-SYNTAX statement. The transfer syntax is identified by means of a unique object identifier.

- Define an access point to the OSI TP services for your UTM application, so that your application can be addressed during communication based on OSI TP.

  The ACCESS-POINT statement is used to specify the address of the access point within the local system, and to assign a symbolic name for addressing the access point in the local UTM application.

  The address defined in ACCESS-POINT must be unique within the UTM application and within the local system (on BS2000 systems for each host). When defining the access point address, you must therefore consult your system or network administrator.

  A partner application that wishes to communicate with the local application on the basis of the OSI TP protocol identifies the local UTM application using the access point address and the network address of your system. The network address of the access point must be specified when generating the remote partner applications.

  > **!** **Unix, Linux and Windows systems**
  > Please note the maximum number of connections that can be established at a time via one access point. For details see **ACCESS-POINT statement** in section "ACCESS-POINT - create an OSI TP access point".

- Define a logical access point (OSI-LPAP partner) for each partner application and the connections between the local application and partner application

  This involves the definition of an OSI-LPAP statement and an OSI-CON statement. An OSI-LPAP statement must be generated for each partner application. The OSI-CON statement defines the connections between the UTM application and the communication partner. The definition is generated via the two access points (in the local application and in the partner application) between which connections are to be established. The OSI-CON statement is used to specify the network address of the remote access point and the name of the local access point (defined in ACCESS-POINT). The OSI-LPAP statement is used to define the number and names of parallel connections (associations) to the partner application. The address of the remote access point must match that of the access point generated in the partner application.

  If connections are to be established automatically as soon as both communication partners are available (i.e. started), this must be specified when generating **both** partners. In openUTM, this is achieved using the statement OSI-LPAP...,CONNECT=.

  The partner started last then establishes the connection. With OSI-LPAP...,CONTWIN=, you can also define the number of connections to the communication partner for which the local application is to act as the contention winner.

- Assign local transaction codes to the services of the partner applications, which are then used to address remote services in the local application

  Each of these transaction codes is defined using an LTAC control statement. The transaction code can be assigned uniquely to the partner application via the LPAP partner using LTAC..., LPAP=*osi-lpapname*. In addition, you must specify the transaction code of the program unit in the partner application using LTAC..., RTAC=. Ask the operator of the partner application for this transaction code. If the name and type of the remote TAC are specified incorrectly, this is not detected by the KDCDEF generation tool, since KDCDEF does not have any information on the configuration of the partner application. The error is not detected until the local application requests this LTAC.

You can also perform the following steps for communication based on OSI TP:

- Define global values for all connections from the application to communication partners

  Using the UTMD statement, you can restrict and define the time spent waiting for confirmation from the communication partner, and specify the total number of jobs submitted to partner applications that can be processed simultaneously in the local application (via OSI TP and LU6.1). By defining appropriate limit values, you can prevent connections from becoming blocked or from being terminated prematurely. You can also ensure that all tasks of the application are occupied by jobs from remote applications. The values defined here also apply for communication based on LU6.1.

- Generate replacement connections

  Replacement connections can be generated by issuing two OSI-CON statements for the same connection. They are used to interact alternatively with various partners without having to take this into consideration in the program units. The two partners may be located in different systems. The replacement connection is generated by assigning two OSI-CON statements with different partner addresses (remote access point addresses) to an OSI-LPAP statement, thereby allocating two different partners to the same logical access point (LPAP partner). However, both connections to the alternative partner applications must not be activated simultaneously. For this reason, ACTIVE=YES may only be specified in one OSI-CON statement. You can switch to the replacement connection to the alternative partner application using the KDCLPAP administration command.

- Define data access control for services of the partner application

  Using the statement LTAC...,LOCK= or LTAC..., ACCESS LIST=, you can secure a partner application service with a lock code. This service is then available locally only to those program units that are running under a user ID (KCBENID) and have been started by a client (KCLOGTER) that have the appropriate authorization.

- Define data access control for services of the local application

  If you wish to restrict a partner application’s access to certain services of the local application, you can secure critical services with a lock code or an access list. With the KSET statement, you can define a key set containing the key codes for services that can be accessed by the partner application. This key set is assigned to the logical access point of the partner application using the statement OSI-LPAP...,KSET=. The partner application can then call only those TACs which are either not secured or for which the partner application has the appropriate authorization.

- Assign administration authorization for TACs of the partner application

  In the OSI-LPAP statement, you can define whether the partner is to be granted administration authorization in the local application. The authorization level is defined using OSI-LPAP...,PERMIT=.

- Assign UTM SAT administration authorization for TACs of the partner application

  Using OSI-LPAP...,PERMIT= you also can define whether the partner is to be granted UTM SAT administration authorization in the local application.

The diagram below summarizes the areas in which the generation of the local application must be coordinated with the generation of the partner application. A standard application context generated by openUTM is used. You must not therefore enter an APPLICATION-CONTEXT statement.

For more information on generating applications with distributed processing, in particular via OSI TP, refer to the example generation „*Comfo*Travel“ in section "Example generation: ComfoTRAVEL".

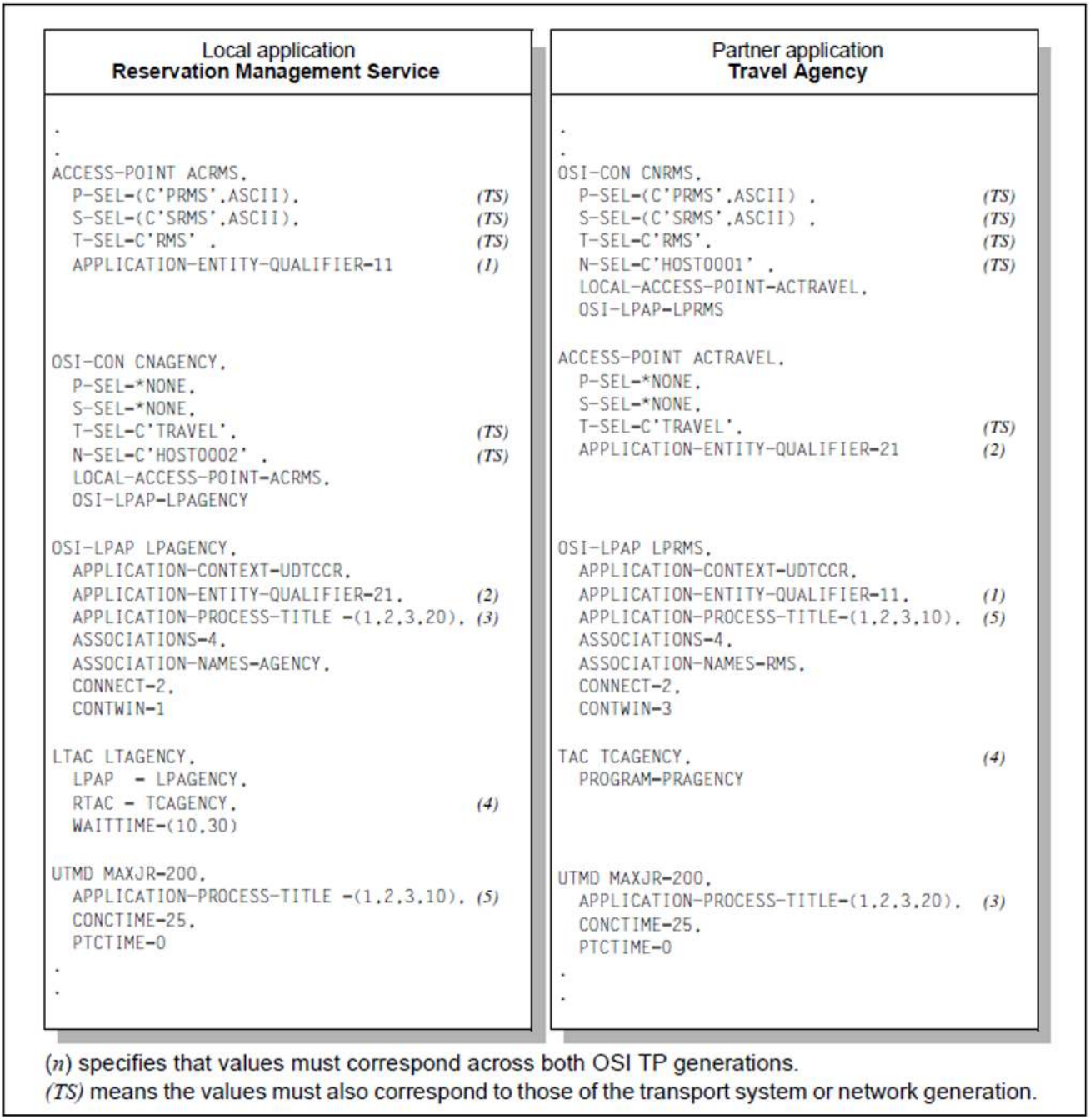


Figure 8: Coordination during the generation of OSI TP applications

3.2.3 OSI-LPAP bundles

OSI-LPAP bundles allow automatic distribution of messages over multiple OSI-LPAP partners. If a UTM application exchanges a large number of messages with a partner application, it may make sense in terms of the load balancing to start several instances of the partner application and distribute the messages among the separate instances. In a OSI-LPAP bundle, openUTM takes care of distributing the messages to the instances of the partner application.To achieve this, the program units in the APRO call must address the MASTER-OSI-LPAP.

One case of an application with this type of message distribution when a UTM application communicates via BeanConnect with a JEE conform application server. If the application server is run as a cluster application, then the messages sent to the application server should be distributed among the separate instances of the cluster (see also the "BeanConnect for openUTM" manual).

A further application scenario is communication from a standalone UTM application to a UTM cluster application. This allows messages to the UTM cluster application to be distributed across the individual node applications.

> Connecting a standalone UTM application to a UTM cluster application via an OSI-LPAP bundle works in the same way as communication between two standalone UTM applications with OSI-LPAP bundles. No special issues related to clusters need to be observed.

An OSI-LPAP bundle consists of one master LPAP and several slave LPAPs. The slave LPAPs are assigned to the master LPAP during generation. In this case, OSI-CONs that belong to different slave LPAPs address the various partner applications.

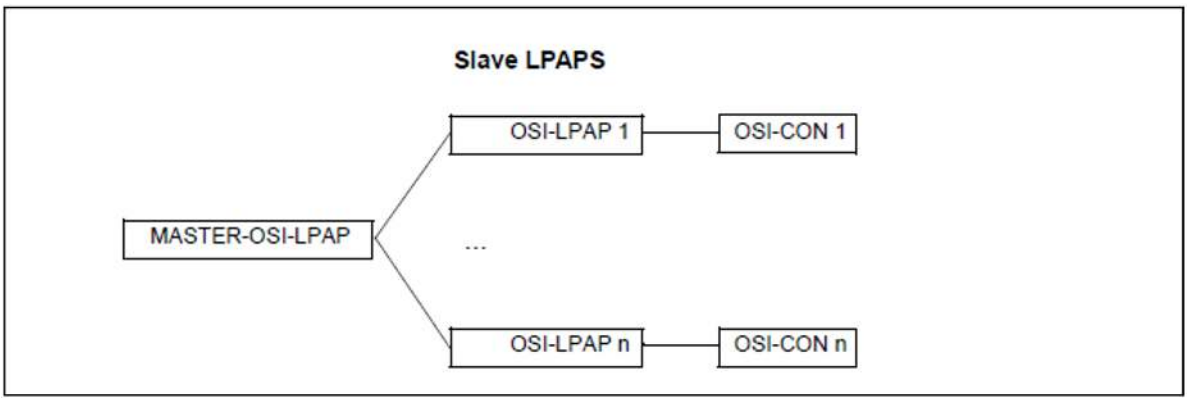


Figure 9: Example of a OSI-LPAP bundle

Generating OSI-LPAP bundles

MASTER-OSI-LPAP statement in section "MASTER-OSI-LPAP - Defining the master LPAP of an OSI-LPAP bundle"
Defines the name and properties of the master LPAP in a OSI-LPAP bundle:

- *master-lpap-name*

  Name for the master LPAP.
- APPLICATION-CONTEXT=

  Application context to be used for the communication with the remote partner.
- STATUS=

  Specifies whether messages can be sent to this LPAP bundle.

OSI-LPAP statement in section "OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP"
The following properties must be specified for the generation of a slave LPAP:

- *lpap-name*

  Name of the slave LPAP.

- BUNDLE=master-lpap-name

  Name of the master LPAP. The master LPAP specified here must be defined in a MASTER-OSI-LPAP statement. If you specify BUNDLE, this OSI-LPAP becomes a slave LPAP of the specified master LPAP.

  `MASTER-OSI-LPAP` *master* `, ...`
  `OSI-LPAP` *slave-lpap* `, BUNDLE=` *master* `, ...`

- APPLICATION-CONTEXT=

  Application context to be used for the communication with the remote partner.

  All slave LPAPs of a LPAP bundle must be assigned to the same application context as the master LPAP.

OSI-CONs of LPAPs in an OSI-LPAP bundle

- Physical connections (OSI-CONs) must not be assigned to master LPAP. This means it may not be specified as an OSI-LPAP in a OSI-CON statement. The master LPAP always uses the connections assigned to the slave LPAPs.
- All OSI-CONs of all slave LPAPs of a LPAP bundle must be assigned to the same local ACCESS-POINT.

Distributing messages

> The following information on distributing messages applies equally to LU6.1 and OSI TP.

Program units can address a slave LPAP as well as a master LPAP with the APRO call. APRO calls to a slave LPAP are not distributed by openUTM. APRO calls to a master LPAP are distributed as follows by openUTM:

- openUTM addresses the slave LPAPs in sequence using APRO calls sent to a master LPAP.
  - openUTM always attempts in this case to find a slave LPAP to which a connection has already been established and, if a queued message is to be sent to the partner (APRO AM), whose queue level has not been reached yet.
  - If the first APRO call to a master LPAP in a transaction is APRO DM, then openUTM only returns the return code 40Z/KD10 when there is no connection to any slave LPAP.
  - If the first APRO call to a master LPAP in a transaction is APRO AM, then openUTM only selects a slave LPAP with a cleared connection when there is no connection yet to any of the slave LPAPs. In this case the connection is initiated for the slave LPAP.
  - When searching for a slave LPAP with an established connection, a connection is initiated for every slave LPAP found that does not have a connection yet.
- All APRO calls sent to the MASTER-LPAP in a single transaction address the same slave LPAP.
  For this reason, an APRO call for a second message to a partner application may be rejected if, for example, the queue level for the slave LPAP has been exceed or the connection has been lost in the meantime.

- Messages that have already been assigned to a slave LPAP are not reassigned in sequence to another slave LPAP any more. Single exception:Asynchronous messages if MAX MOVE-BUNDLE-MSGS=YES was generated. (default is NO).

  If the connection for dialog messages is lost after the APRO call, then the dialog message is rejected just like for "normal" LPAPs and the transaction is rolled back, if necessary.

Information displayed in the KB header

> The following information on the display in the KB header applies equally to LU6.1 and OSI TP.

In services started for received messages, openUTM always displays the name of the LTERM or (OSI-)LPAP through which the message was received in the KB header.

The following therefore applies for LPAP bundles:
In services started for messages received through a slave LPAP, the name of this slave LPAP is displayed in the KB header and **not** the name of the master LPAP.

With INIT PU you can obtain information on whether the (OSI-)LPAP in the KB header is the slave LPAP of an LPAP bundle as well as the name of the master LPAP.

3.3 Coordinating the UTM and BCAM configurations (BS2000 systems)

During distributed processing via LU6.1 and OSI TP network connections are required to enable applications to communicate with each other. openUTM uses the services of the BCAM transport system for these network connections.

To allow you to establish these network connections, addresses must be assigned to both communication partners. These addresses must be unique throughout the network. The network type required for communication is defined by means of appropriate entries in KDCDEF generation (BCAMAPPL T-PROT= statement) and in BCAM generation. It is important to note that openUTM does not distinguish between ISO and TCP/IP networks (ISO and RFC1006 entries for the T-PROT parameter are synonymous). This distinction must be made in BCAM generation when defining connections between participating computers.

In the case of a connection via RFC1006, BCAM uses the number 102 as the partner’s listener port number by default. If a different port number is to be used, e.g. because the transport system of the partner cannot use port number 102, then the port number can be specified via the LISTENER-PORT operand in the PTERM and (OSI-) CON statements.

3.4 Providing address information for the CMX transport system (Unix, Linux and Windows systems)

In the case of distributed processing via LU6.1 or OSI TP and when connecting clients of type PTYPE=APPLI and UPIC-R, openUTM uses the CMX transport access system (Unix sytems and Windows systems). When communicating using CMX, the connection is established using TCP/IP-RFC1006.

You must provide the address information for the transport system via the UTM generation.

> ¡ With openUTM V6.5, the TNS functionality was supported for the last time! As of openUTM V7.0 it is necessary to store the complete address information with the UTM generation.

Port number 102 for TCP/IP connections

The applications are accessed via port numbers in connections via TCP/IP-RFC1006.
Please note that using port number 102 for local transport system endpoints (BCAMAPPL, ACCESS-POINT) in UTM applications on Unix, Linux, and Windows systems may require additional privileges (for example, root).

3.4.1 Providing address information with KDCDEF

The address information is stored in the PRONAM, N-SEL, LISTENER-PORT operands, as well as in the T-PROT and TSEL-FORMAT operands. You must note the following:

- You must specify the TCP/IP host name for PRONAM or N-SEL, see "Specifying computer names in KDCDEF generation". For actively established connections, openUTM determines the name from the IP address, and for passively established connections the IP address from the name.
- You should enter a port number for LISTENER-PORT. The port number must always match the port number used by the communication partner.
- You must specify RFC1006 for T-PROT.
- It is recommended to enter data for the TSEL-FORMAT operand. If you do not specify anything there, then KDCDEF assigns one of the following values, depending on the OPTION statement:
  - for CHECK-RFC1006=YES, a default value based on the set of characters in the name of the corresponding application or partner
  - for OPTION CECK-RFC1006=NO, the invalid value 'U' or '?' (= undefined)

OSI TP connection

In the following example, port number `10000` is used in the local application and port number `12000` is used in the remote application. The remote application is running on the computer named `CENTRAL1`.

```
ACCESS-POINT BSPOSITP -
,LISTENER-PORT=10000 -
,T-PROT=RFC1006 -
,TSEL-FORMAT=T -
,P-SEL=.., ,S-SEL=... ,T-SEL=... -
,...
```

```
OSI-CON OSICON01 -
,LOCAL-ACCESS-POINT=BSPOSITP -
,LISTENER-PORT=12000 -
,T-PROT=RFC1006 -
,N-SEL=CENTRAL1 -
,P-SEL=... ,S-SEL=... ,T-SEL=... -
,...
```

The specifications for P-SEL, S-SEL and T-SEL need to match the values specified in the generation of the partner application (see the sample in section "Generation procedure for distributed processing based on OSI TP").

LU6.1 connection

In the following example, port number `10010` is used in the local application and port number `12010` is used in the remote application. The remote application is running on the computer named `CENTRAL2`.

```
BCAMAPPL BSPLU61 -
,LISTENER-PORT=10010 -
,T-PROT=RFC1006 -
,T-SEL-FORMAT=T -
,...
```

```
CON LU61PART -
,BCAMAPPL=BSPLU61 -
,PRONAM=CENTRAL2 -
,LISTENER-PORT=12010 -
,T-PROT=RFC1006 -
,TSEL-FORMAT=T -
,...
```

PTYPE=APPLI connection

In the following example, port number `10020` is used in the local application and port number `12020` is used in the remote application. The remote application is running on the computer named `CENTRAL3`.

```
BCAMAPPL BSPAPPLI -
,LISTENER-PORT=10020 -
,T-PROT=RFC1006 -
,T-SEL-FORMAT=T -
,...
```

```
PTERM APPLPART -
,PTYPE=APPLI  -
,BCAMAPPL=BSPAPPL -
,PRONAM=CENTRAL3 -
,LISTENER-PORT=12020 -
,T-PROT=RFC1006 -
,TSEL-FORMAT=T -
,...
```

PTYPE=UPIC-R connection

In the following example, port number `10030` is used in the local application. The remote application is running on the computer named `CENTRAL4`.

```
BCAMAPPL BSPUPR -
,LISTENER-PORT=10030 -
,T-PROT=RFC1006 -
,T-SEL-FORMAT=T -
,...
```

```
PTERM UPRPART -
,PTYPE=UPIC-R -
,BCAMAPPL=BSPUPR -
,PRONAM=CENTRAL4 -
,T-PROT=RFC1006 -
,TSEL-FORMAT=T -
,...
```

In the KDCDEF call consistency checks are performed on the address information. The checks are performed because the OPTION statement uses the operand CHECK-RFC1006=YES

3.4.2 Converting address information from TNS entries to KDCDEF

If you have been providing the address information to date using TNS entries, then you have only entered in the UTM generation the application name, possibly the host name of the communication partner, and the name of your UTM application. You must specify these names in TNS as GLOBAL NAMES. The host name/IP address associations are determined by the TNS entry.

When converting from TNS entries to KDCDEF, the only information you still need to specify is the port number.

It is recommended that you always administer the full generation information for a UTM application centrally. To do this, proceed as follows:

- Include the address information of the TNS entries that are of relevance for the UTM application in the KDCDEF generation.
- Delete these TNS entries, including those from the TNS directory.

Based on the LU6.1 connection from section "Providing address information with KDCDEF" the following presents an example of the changes you will need to make when converting:

LU6.1 connection

In the following example, port number `10010` is used in the local application `BSPLU61` and port number `12010` is used in the remote application `LU61PART`. The remote application is running on the computer named `CENTRAL2`.

Before conversion

KDCDEF:	`BCAMAPPL BSPLU61`
	
	`CON LU61PART -`
	`,BCAMAPPL=BSPLU61 -`
	`,PRONAM=CENTRAL2`
	`,...`

TNS entries:	`BSPLU61\`
	`  TSEL RFC1006 T'BSPLU61'`
	`  TSEL LANINET A'10010'`
	
	`LU61PART.CENTRAL2\`
	`  TA RFC1006 address of CENTRAL2 PORT 12010 T'LU61PART'`

After conversion

When converting you must specify

- for the local application name
  the port number in the LISTENER-PORT parameter of the KDCDEF statement BCAMAPPL instead of the port number in the TNS entry (TSEL LANINET).
- for the remote partner
  the processor name of the partner computer in the PRONAM parameter and the port number in the LISTENER-PORT parameter of the KDCDEF statement CON instead of the address and port number in the transport address of the TNS entry (TA).

In order to be able to determine the IP address of the partner computer, the name CENTRAL2 must be known to the DNS.

KDCDEF:	BCAMAPPL BSPLU61 - ,LISTENER-PORT=10010 - ,T-SEL-FORMAT=T - ,... CON LU61PART - ,BCAMAPPL=BSPLU61 - ,PRONAM=CENTRAL2 - ,LISTENER-PORT=12010 - ,TSEL-FORMAT=T - ,...

LU6.1 connection using symbolic names

In the following example, port number 10010 is used in the local application BSPLU61 and port number 12010 is used in the remote application LU61PART. The remote application is running on the computer named CENTRAL2.

Before conversion

KDCDEF:	BCAMAPPL LU61 CON LU61 - ,BCAMAPPL=LU61 - ,PRONAM=PROLU61 ,...
TNS entries:	LU61\ TSEL RFC1006 T'BSPLU61' TSEL LANINET A'10010' LU61.PROLU61\ TA RFC1006 ***address of CENTRAL2*** PORT 12010 T'LU61PART'

After conversion

In contrast to the example “LU6.1 connection”, you must also change the BCAMAPPL name and the CON name in the TSEL parameters of the TNS entries. In this case this means you must use the real names instead of the symbolic names.

When converting you must

- change the name of the local application
  in the BCAMAPPL parameter to the value you used in the local TNS entry (TSEL RFC1006).
- for the local application name
  as in the example “LU6.1 connection”, specify the port number in the LISTENER-PORT parameter of the KDCDEF statement BCAMAPPL instead of the port number in the TNS entry (TSEL LANINET).
- the name of the remote application
  change the value in the CON statement to the value you used in the remote TNS entry (TA).

- for the remote partner
  as in the example “LU6.1 connection”, the processor name of the partner computer in the PRONAM parameter and the port number in the LISTENER-PORT parameter of the KDCDEF statement CON instead of the address and port number in the transport address of the TNS entry (TA)

In order to be able to determine the IP address of the partner computer, the name `CENTRAL2` must be known to the DNS.

KDCDEF:	`BCAMAPPL BSPLU61 -`
	`,LISTENER-PORT=10010 -`
	`,T-SEL-FORMAT=T -`
	`,...`
	
	`CON LU61PART -`
	`,BCAMAPPL=BSPLU61 -`
	`,PRONAM=CENTRAL2 -`
	`,LISTENER-PORT=12010 -`
	`,TSEL-FORMAT=T -`
	`,...`

3.5 Providing address information for the SOCKET transport system (Unix, Linux and Windows systems)

In connections to a TS application with PTYPE=SOCKET, communication is performed via the socket interface using native TCP/IP as the transport protocol. For these socket applications the address information will be provided for the transport system via the UTM generation.
The address information is stored in the PRONAM and LISTENER-PORT operands as well as in T-PROT and TSEL-FORMAT. In addition, the BCAMAPPL operand is also important. Note the following points in this context:

- For PRONAM you must specify the TCP/IP host name, see "Specifying computer names in KDCDEF generation" . openUTM determines the IP address from it.
- You must enter a port number for LISTENER-PORT. The port number absolutely must match the port number used by the communication partner. Specification of a LISTENER-PORT is mandatory.
- You must specify SOCKET for T-PROT.
- It is recommended to set a value in the TSEL-FORMAT operand.
- In BCAMAPPL you must specify an application name for which T-PROT=SOCKET was generated.

> **!** **Unix, Linux and Windows systems**
> Please note the maximum number of connections that can be established at a time via one transport system end point. For details see **BCAMAPPL statement** in section "BCAMAPPL - define additional application names".

PTYPE=SOCKET connection

In the following example, port number `10010` is used in the local application and port number `12100` is used in the remote application `LU61PART`. The remote application is running on the computer named `CENTRAL5`.

```
BCAMAPPL BSPSOC -
,LISTENER-PORT=10100 -
,T-PROT=SOCKET -
,T-SEL-FORMAT=T -
,...

PTERM SOCPART -
,PTYPE=SOCKET -
,BCAMAPPL=BSPSOC -
,PRONAM=CENTRAL5 -
,LISTENER-PORT=12100 -
,T-PROT=SOCKET -
,TSEL-FORMAT=T -
,...
```

3.6 Network connection (Unix, Linux and Windows systems)

During distributed processing, the UTM application is connected to the network via network processes. These processes are responsible for handling connection setup requests and for managing data transfer via the connection.

The main process *utmmain* of a UTM application starts one or more network processes, which can in turn establish and manage numerous connections. Assignments between connections and processes are controlled through the use of **listener IDs**. All connections with the same listener ID are managed by threads of the same network process.

There are particular network process types for CMX and socket connections. CMX connections use a process type called *utmnet*, and socket connections use *utmnets or utmnetssl*.

Listener IDs can be defined in the **ACCESS-POINT statement** for access points and in the **BCAMAPPL statement** for application names:

- LISTENER-ID=number

  This assigns a listener ID to the access point or application name as administrative information.

  Due to the different types of net processes, the listener IDs for CMX connections and the listener IDs for Socket or SSL Socket connections are separate value ranges.

The connection is assigned to a network process by means of the listener ID allocated to the local application name or the local access point. openUTM assigns the listener ID 0 to all application names and access points for which a listener ID has not been explicitly defined. All of these connections are then served by a single network process.

Unix-, Linux- und Windows-Systeme

Please note that only a maximum of 1000 connections can be established per network process at a time. If you need moreconnections in your application, you must define several local application names e.g. local access points. For Details see **BCAMAPPL statement** in section "BCAMAPPL - define additional application names" e.g. **ACCESS-POINT statement** in section "ACCESS-POINT - create an OSI TP access point".

3.7 Computer names (Unix, Linux and Windows systems)

In openUTM, computer names up to 64 characters can be specified in the KDCDEF generation. The KDCNAMEINFO tool is also available for determining the fully qualified computer name from the IP address.

3.7.1 Specifying computer names in KDCDEF generation

When specifying the computer name, observe the following points:

- In a “Fully Qualified Domain Name” (FQDN), if the section up to the first full stop (known as the “host name”) is longer than eight characters, you have to specify the FQDN for PRONAM or N-SEL.
- In all other cases, you can choose whether you specify the FQDN for PRONAM or N-SEL or just the section up to the first dot, known as the “host name”.
- We recommend using the KDCNAMEINFO utility program to determine the host name (see "KDCNAMEINFO tool").
- The FQDN-name may be a maximum of 64 bytes long, so that you can specify it in the configuration of an UTM application.

3.7.2 KDCNAMEINFO tool

The KDCNAMEINFO tool determines the associated computer name for an IPv4 or IPv6 address and outputs it to *stdout*. You specify this computer name when generating your UTM application as the value for the parameter PRONAM or N-SEL. You can thus ensure that the specified host name / processor name in the UTM generation on a Unix, Linux or Windows system matches the network configuration.

Calling KDCNAMEINFO

The tool is called as follows:

utmpath`/ex/kdcnameinfo` *ip_address* (Unix and Linux systems) or

utmpath`\ex\kdcnameinfo` *ip_address* (Windows systems)

ip_address is the IPv4 or IPv6 address of the partner system, with which the UTM application wants to communicate. The following applies:

- *ip_address* for IPv4 is specified using the traditional dot notation:

  `xxx.xxx.xxx.xxx`

  The individual xxx blocks are limited to three decimal places each. The content of the blocks is interpreted as a decimal number.
- *ip_address* for IPv6 is specified using the conventional colon notation:

  `x:x:x:x:x:x:x:x`

  Each x represents a hexadecimal number between 0 and FFFF. The alternative notations for IPv6 addresses are permitted (see RFC2373).

  If the IPv6 address contains an embedded IPv4 address in dot notation, the above rules apply to the blocks in the IPv4 address, i.e. the blocks in the IPv4 address are then interpreted as a decimal number.

4 Generating selected objects and functions of the application

This chapter describes how to configure certain objects of your UTM application and explains which KDCDEF control statements or which of the individual operands are relevant for describing the objects. This applies to the following UTM objects:

- Clients ("Connecting clients to the application")
- Printers, printer control LTERMs, printer pools on BS2000, Unix and Linux systems ("Generating printers (on BS2000, Unix and Linux systems)") as well as RSO printers connected to a UTM application on BS2000 systems ("Generating RSO printers (BS2000 systems)")
- Service-controlled queues ("Generating service-controlled queues")
- Message modules ("UTM messages")
- Multiplex connections of a UTM application on BS2000 systems ("Message distribution and multiplexing with OMNIS (BS2000 systems)")
- BLS load modules and common memory pools of a UTM application on BS2000 systems ("Generating load modules, common memory pools and shared code (BS2000 systems)")

In addition, several selected UTM functions are described here that affect KDCDEF control functions and whose operands are significant to the use of these functions. This is true for the following functions:

- Code conversion ("Code conversion")
- Job control using priorities and process constraints ("Code conversion")
- Access control functions ("Data access control")
- Encryption of messages on connections to clients ("Message encryption on connections to clients")
- Coupling of resource managers and databases ("Defining database linking")
- Internationalization of a UTM application on BS2000 systems ("Internationalizing the application - XHCS support (BS2000 systems)")
- System access control using Kerberos on BS2000 systems ("Defining system access control")

See also the openUTM manual “Concepts und Functions”.

4.1 Connecting clients to the application

This section describes the generation of terminals, UPIC clients, HTTP clients, transport system applications and OpenCPIC clients. Transport system applications are DCAM, CMX and socket applications as well as UTM applications that are generated as transport system applications. These will subsequently be referred to as TS applications.

The options that the UPIC clients offer are described in detail in the manual „openUTM-Client for the UPIC Carrier System”.

Each client that wants to use the services of a UTM application must be known to the UTM application. A client is known to a UTM application if it is assigned to a logical connection point defined in the configuration. There are various different types of client:

- For terminals, UPIC clients and TS applications, a logical connection point is known as an **LTERM partner**. There are two methods of connecting to an LTERM partner:
  - You generate the client for an individual connection by defining the physical client using a PTERM statement and then assigning an exclusive LTERM partner, see below. A client must always be generated with PTERM, when connections to this client are to be established in the UTM application (e.g. to TS applications). You only need to issue a PTERM statement to other clients when you want to assign the other client a specific logical property, e.g. special access rights that you do not want to assign to any LTERM pool.
  - You define a pool of LTERM partners, also called an LTERM pool, see "LTERM pools". You can connect several clients using the LTERM pool.
- For OpenCPIC clients, the logical connection point is known as the **OSI LPAP partner**. It is possible to establish several parallel connections via an OSI LPAP partner.

The first two sections show the basic steps required to connect a client. The section "Defining the client sign-on services" through section "Examples of the generation of a client/server cluster" go into more detail on certain topics, including the signing-on process, security functions and addressing.

4.1.1 Connecting clients via LTERM partners

If you wish to connect terminals, UPIC clients and TS applications individually, you will need to supply the following generation statements for each client:

- an LTERM statement for the logical connection point
- a PTERM statement for the physical client.

UPIC clients and TS applications may also require a BCAMAPPL statement. Limits, maximum values and parameters, which are to be set throughout the application for communication between clients and the UTM application, are defined in the MAX statement.

LTERMs and PTERMs may also be created dynamically (objects KC_LTERM and KC_PTERM). Moreover, the assignment of the client to the LTERM partner in the PTERM statement can be adapted dynamically at a later stage using administration functions. For example, you can assign another client (of the same type) to an LTERM partner during operation, or assign another LTERM partner – for which you may have defined different access rights – to a client. See also the openUTM manual “Administering Applications”.

LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
The most important properties for LTERM partners, via which clients can connect to an application, are defined with the following operands:

- *ltermname*

  Name of the LTERM partner. Logical name via which the client, to which the LTERM partner is assigned, is addressed by the program units of the application.

- KSET=

  Key set of the LTERM partner, i.e. an authorization profile that defines which parts of the application program (which TACs) are available to the client connecting to the application via this LTERM partner.

- LOCALE= (only BS2000 systems)

  LTERM-specific language environment of the clients that connect to the application via this LTERM partner. This language environment is also used by openUTM to output messages, as long as no user is signed on.

- LOCK=

  Lock code as system access control. The connection is only established when the client signs on to openUTM with a user ID, for which a key set was generated, using a key code corresponding to this lock code.

- USAGE=D

  Type of communication partner. In this case, dialog partners are connecting to the application via the LTERM partner. Messages can be exchanged in both directions.

- USER=

  The user ID under which the client is automatically signed on when a connection has been established, see "Automatic sign-on under a specific user ID". You are also able to define other characteristics for this user ID, see "Generating security functions".

PTERM statement in section "PTERM - define the properties of a client/printer and assign an LTERM partner"
The most important properties for physical clients are defined with the following operands:

- *ptermname*

  Name of the client as generated in the system of the server application.

  BS2000 systems (without Socket application):
  The BCAM name of the client must be specified.

  Socket applications:
  If the connection is to be established from the local UTM application to the client, then any *ptermname* can be selected. Otherwise, the name must have the format PRT*nnnnn*.Here *nnnnn* means the port number used by the socket application to establish the connection.

  See "Providing address information" for more information.

- BCAMAPPL=

  Name of the local application via which the transport system establishes the connection between the client and the UTM application. This name must be defined in a BCAMAPPL statement or using MAX ...APPLINAME=. If you omit this operand, the name is taken from MAX ...APPLINAME=. Terminals may only use names that are defined in MAX ... APPLINAME=.

- ENCRYPTION-LEVEL=

  For UPIC clients you specify the minimum encryption level that must be maintained on the connection to the client. You can specify trustworthy for the client, which means that this client is permitted to work with the UTM application without encryption. See also “Message encryption on connections to clients" for more information on encryption.

- LTERM=

  The LTERM partner *ltermname*, via which the client connects to the UTM application, is assigned to the physical client as a logical connection point.

- PRONAM=

  Symbolic name of the processor on which the client resides.

- PTYPE=

  Type of client connected via the LTERM partner. Here you specify whether the client is a transport system application, a UPIC client or a terminal.

- T-PROT=, TSEL-FORMAT= (only on Unix, Linux and Windows systems),LISTENER-PORT= (with PTYPE=SOCKET also on BS2000 systems)

  Components of the transport address of a remote UPIC client or a TS application see "Providing address information".

- USAGE=D (only BS2000 systems)

  USAGE=D defines that the communication partner is a dialog partner. Messages can be exchanged between the UTM application and the client.

- USP-HDR=

  With a sockets applications that use the USP protocol, this parameter controls which of the output messages openUTM is to create a protocol header for, see "USP headers for outputmessages to socket connections".

BCAMAPPL statement in section "BCAMAPPL - define additional application names"
It is possible to define additional application names for UPIC clients and TS applications.

- *appliname*

  Name of the local application used by the transport system to establish the connection between the client and the UTM application. If this name is used for socket applications, it may not be used by a different type of partner.

- SIGNON-TAC=

  Specifies if and when a sign-on service takes place, when a client attempts to sign on under this application name, see "Generating sign-on services for clients".

- TSEL-FORMAT=, LISTENER-ID= (only on Unix, Linux and Windows systems),LISTENER-PORT= (with PTYPE=SOCKET also on BS2000 systems)
  T-PROT=

  For information about the components of the transport address under which client contacts the UTM application, see "Providing address information".

MAX statement in section "MAX - define UTM application parameters"
Default and maximum values, which are relevant for communication of clients with the UTM application, are defined with the following operands:

- CONN-USERS=

  Controls the utilization of the application. The operand defines the maximum number of users who can simultaneously work with the application. In the case of an application for which no user IDs are generated, CONN-USERS= defines the maximum number of clients that can simultaneously connect to the application via LTERM partners.

- LOCALE= (on BS2000 systems only)

  Defines the default language environment (locale) of the UTM application. The locale generated here is assigned to the clients connected via LTERM partners or LTERM- pools as the default value for the language environment. The default setting applies unless a specific locale is defined for these objects in the corresponding LTERM or TPOOL statements. See also "Defining the language environment – setting the locale".

4.1.2 LTERM pools

A particular number of LTERM partners with the same logical properties are defined for an LTERM pool as logical connection points for clients. Clients with the same technical properties (partner and processor type) can connect to a UTM application via these LTERM partners. The assignment only applies for the duration of a session; there is no static assignment between a client and an LTERM partner.

An LTERM pool must be configured in a TPOOL statement (in place of LTERM/PTERM statements). In addition, in the same way as for a single connection, a BCAMAPPL statement may be necessary, see "Connecting clients via LTERM partners". The settings in the MAX statement are also valid for LTERM pools, see "Connecting clients via LTERM partners".

Various types of LTERM pools can be configured:

- LTERM pools via which only clients of a particular type (PTYPE=), located on a particular processor (PRONAM=), can connect to a UTM application.
- LTERM pools, via which clients of a particular type can connect to a UTM application, regardless of the processor on which they reside (open LTERM pools).

In UTM applications on BS2000 systems you can also generate the following types of LTERM pools:

- LTERM pools for all terminals regardless of the terminal type, yet located on a particular processor.
- LTERM pools for all terminals regardless of the terminal type and regardless of type of the computer on which it is located.

TPOOL statement in section "TPOOL - define an LTERM pool"
The most important properties for LTERM pools are defined with the following operands:

- BCAMAPPL=

  Name of the local application via which the transport system establishes the connection between the client and the UTM application. The name must be defined in a BCAMAPPL statement or using MAX ... APPLINAME=.
- CONNECT-MODE=

  With CONNECT-MODE= you specify if a UPIC client or TS application may connect to the application multiple times under the same name via the LTERM pool.
- KSET=

  Key set of the LTERM pool that uses key codes to define the access rights of the clients which connect to the UTM application via the LTERM pool.
- USER-KSET=

  In UTM applications with user IDs the USER-KSET key set for UPIC clients and TS applications specify limited system access rights (in comparison with KSET). The key set in USER-KSET takes effect when the client does not pass a user ID to openUTM while establishing the connection/conversation or while in the sign-on service.
- LOCK=

  System access control of the LTERM pool, i.e. lock code assigned for all LTERM partners of the pool. The connection is only established if the client signs on to openUTM with a user ID whose key set has the corresponding key code.

- ENCRYPTION-LEVEL=

  For UPIC clients you specify the minimum encryption level that must be maintained on the connection to the client. You can specify trustworthy for the client. See also "Message encryption on connections to clients" for more information on encryption.

- LTERM=

  LTERM prefix from which unique LTERM partner names are created with *number* LTERM partners of the LTERM pool.

- NUMBER=

  Number of LTERM partners configured for this LTERM pool. This also implicitly defines the maximum number of clients that can connect to this LTERM pool at the same time.

- PRONAM=

  Name of processor which must contain the client connected via the LTERM pool.

- PROTOCOL=

  Specifies if the user services protocol is used or not.

- PTYPE=

  Type of client connected via the LTERM pool.

- LOCALE=

  LTERM-specific language environment that applies to all clients which connect to the application via LTERM partners. This language environment is also used byopenUTM to output messages, as long as no user is signed on.

LTERM pools and subnets

IP address ranges can be defined using the SUBNET statement. These address ranges are referred to as subnets. Each subnet can be assigned to an LTERM pool using a TPOOL statement. All clients which set up a connection to the UTM application from a subnet defined in this way are then allocated to this LTERM pool.

SUBNET statement in section "SUBNET - define IP subnets"
The properties for subnets are defined with the following operands:

- mapped-name

  Local name of the subnet. This name must be specified for PRONAM in the TPOOL statement.

- BCAMAPPL=

  Name of the local application to which the client can establish the connection to the UTM application. This name must be defined in a BCAMAPPL statement or with MAX ...APPLINAME= and specified in the TPOOL statement with BCAMAPPL=.

  The client must use this local application name to connect to the UTM application using a TPOOL linked to a SUBNET statement. This assigns it directly to the subnet and the associated LTERM pool. In this case no name resolution via the DNS takes place.

  When a client from the subnet uses a different local application name for connection setup, the client's host name is determined via DNS, and the host name thus ascertained is used for the assignment to the generated partner.

- IPV4-SUBNET= or IPV6-SUBNET=

  Defines the IPv4 or IPv6 subnet.

Assignment of client to an LTERM pool

For clients that want to connect to an application via an LTERM pool, please note that openUTM only assigns a client to one LTERM pool. When selecting the LTERM pool, openUTM considers it more important to match the processor name than the client type.

The table below shows the sequence in which openUTM attempts to assign a client to the generated PTERMs and LTERM pools. The grayed out rows in the table represent LTERM pools that can only exist in a UTM application on BS2000 systems. but not on Unix, Linux and Windows systems.

Assignment of a client	KDCDEF statements: definition of client		Remark
1.	`PTERM`	`PTYPE=partnertype` `PRONAM=processorname`	
2.	`TPOOL`	`PTYPE=partnertype` `PRONAM=processorname`	
3.	`TPOOL`	`PTYPE=*ANY` `PRONAM=processorname`	*only on BS2000 systems*
4.	`TPOOL`	`PTYPE=partnertype` `PRONAM=*ANY`	
5.	`TPOOL`	`PTYPE=*ANY` `PRONAM=*ANY`	*only on BS2000 systems*

1. When establishing a connection, it is first of all checked whether a PTERM statement exists for the client.
   UTM first searches for a PTERM-entry using the short processor name of up to 8 characters as supplied by the transport system in the connection request. If no matching PTERM-entry can be found, then UTM retrieves the long processor name of the communication partner from the transport system which can be up to 64 characters long, and searches for a PTERM-entry with this long processor name.
   A client that was explicitly generated with a PTERM statement cannot connect to a UTM application via an LTERM pool.
2. If an LTERM pool is generated for the processor name (PRONAM) and type (PTYPE) of a client, this client is assigned to this LTERM pool.
   PRONAM can also be the mapped name of a SUBNET statement if a subnet is generated for the IP address of the client.
3. (only on BS2000 systems) If there is no LTERM pool with the processor name and type of the client, the client is assigned to the LTERM pool with the same processor name and PTYPE=*ANY.

   UTM first performs steps 2 and 3 using the short processor name of up to 8 characters as supplied by the transport system in the connection request. If in neither one of the steps 2 and 3 a matching LTERM pool can be found, then UTM retrieves the long processor name of the communication partner from the transport system and repeats the search of steps 2 and 3 with this long processor name.

4. If no such LTERM pool exists, the client is assigned to an “open” LTERM pool with the same type and PRONAM=*ANY, i.e. all clients of a type can connect to the UTM application, regardless of the processor on which they reside.

5. (only on BS2000 systems) If no such LTERM pool exists either, the client is assigned to an LTERM pool with PTYPE=*ANY and PRONAM=*ANY.

If there is no LTERM pool of this type or if the capacity of the LTERM pool selected by openUTM is exceeded, the connection setup request is rejected.

4.1.3 LTERM bundle

With a LTERM bundle (connection bundle) you distribute queued messages to a logical partner application equally among several parallel connections. The logical partner application can comprise several instances of the partner application (e.g. a UTM cluster application). This type of distribution may make sense when a UTM application sends a very large number of queued messages to a partner application, possibly leading to the overloading of one transport connection.

You define a LTERM bundle using LTERM and PTERM statements as already described in the section "Connecting clients via LTERM partners". The following text describes the additional points you must note to work with LTERM bundles.

A LTERM bundle consists of one master LTERM and several assigned slave LTERMs. The slave LTERMs, which must be assigned using PTERM with PTYPE=APPLI or PTYPE=SOCKET, are assigned to a master LTERM through generation.

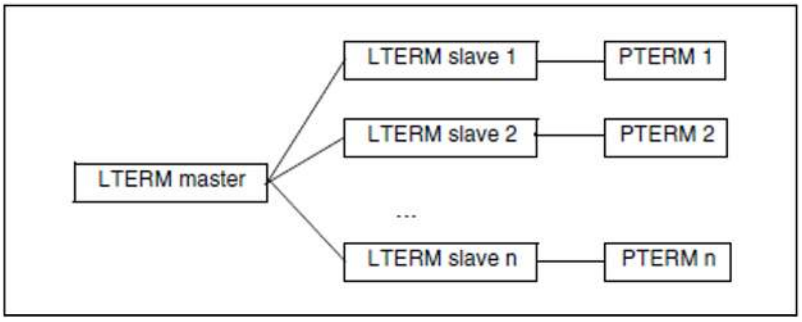


Figure 10: Example of a LTERM bundle

FPUT/DPUT calls

FPUT calls sent by program units to the master LTERM are assigned to one of the slave LTERMs at the end of the transaction:

- openUTM first attempts to find a slave LTERM whose PTERM is connected. If openUTM cannot find such a connection, then it searches for a slave LTERM that was generated with RESTART=YES.

  If openUTM finds a slave LTERM, then all queued messages sent in this transaction to this master LTERM are assigned to the slave LTERM.
- If openUTM cannot find such a slave LTERM, then all asynchronous messages sent to the master LTERM are rejected.
- If a slave LTERM is generated with RESTART=NO and the connection is cleared or lost, then all messages pending output on this LTERM are rejected.

Program units can also send FPUT and DPUT calls directly to the slave LTERMs. However, these messages are not subject to the distribution algorithm described above.

Information displayed in the KB header

Messages can also be received through the slave LTERMs of a LTERM bundle. In services started for received messages, openUTM always displays the name of the LTERM through which the message was received in the KB header. The following therefore applies for LPAP bundles:
In services started for messages received through a slave LTERM, the name of this slave LTERM is displayed in the KB header and **not** the name of the master LTERM.

With the aid of the KDCS call INIT PU you can obtain information on whether the LTERM in the KB header is the slave of an LTERM bundle as well as the name of the master LTERM (see the openUTM manual „Programming Applications with KDCS”).

LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
In addition to the properties for LTERM partners already listed (see "Connecting clients via LTERM partners"), the following operands must also be specified for LTERM bundles:

- BUNDLE=

  Specifies the corresponding master LTERM in the definition of a slave LTERM. The master LTERM specified her must have been generated in a preceding LTERM statement:

  ```
  LTERM master , ...

  LTERM slave1 , BUNDLE= master , ...
  LTERM slave2 , BUNDLE= master , ...

  PTERM slave1 , LTERM= slave1 , PTYPE=APPLI|SOCKET, ...


  PTERM slave2 , LTERM= slave2 , PTYPE=APPLI|SOCKET, ...
  ```

- RESTART=

  Determines how queued messages are handled when the connection to the client is cleared. Messages pending output on a LTERM that were generated with RESTART=NO may be rejected if necessary (see “FPUT/DPUT calls”).

i All LTERM parameters of slave LTERMs except for *ltermname*, USER, QAMSG, RESTART, and STATUS must match the same parameters of the master LTERM. Otherwise they will be overwritten by KDCDEF using the data specified in the master LTERM. No message is output in this case.

When assigning the asynchronous messages to a slave LTERM at the end of a transaction, the QAMSG and RESTART settings are evaluated on the slave LTERM.

All slave LTERMs in a LTERM bundle should be generated identically. KDCDEF does not check this, though.

PTERM statement in section "PTERM - define the properties of a client/printer and assign an LTERM partner"
In addition to specifying the properties for physical clients already listed (see "Connecting clients via LTERM partners"), the following operands must also be specified for the PTERMs assigned to the slave LTERMs in a LTERM bundle:

- PTYPE=APPLI | SOCKET

  All PTERMs in a LTERM bundle must be generated with PTYPE=APPLI or PTYPE=SOCKET. The same PTYPE must be specified here for all PTERMs in a LTERM bundle.

- USAGE=D (BS2000 systems only)

  All PTERMs in a LTERM bundle must be generated with USAGE=D.

All PTERMs in a LTERM bundle should address the same partner application or a partner application of the same type. KDCDEF does not check this, though.

4.1.4 LTERM groups

In a LTERM group you assign one or more LTERMs a connection. The use of LTERM groups is of value if a UTM application is to send queued messages to different partner applications depending on which functional area the message belongs to. In this case, a separate LTERM must be assigned to each functional area in the partner application.

The program units direct their FPUT and DPUT calls to the appropriate LTERM depending on the function. If the partner application to function relationship is 1:1, then each LTERM is assigned one PTERM. If the partner application to function relationship is n:1 and the assignment may change in some cases, then n LTERMs are assigned to one PTERM.

An LTERM group consists of one or more alias LTERMs, called the group LTERMs, and one primary LTERM. You define the group LTERMs using LTERM statements as described in the section "Connecting clients via LTERM partners". Do not assign a PTERM to a group LTERM.

The primary LTERM must be a normal LTERM or the master LTERM of a LTERM bundle. If the primary LTERM is a normal LTERM, then a PTERM with PTYPE=APPLI or PTYPE=SOCKET must be assigned to it. You define the primary LTERM as described in the section "Connecting clients via LTERM partners".

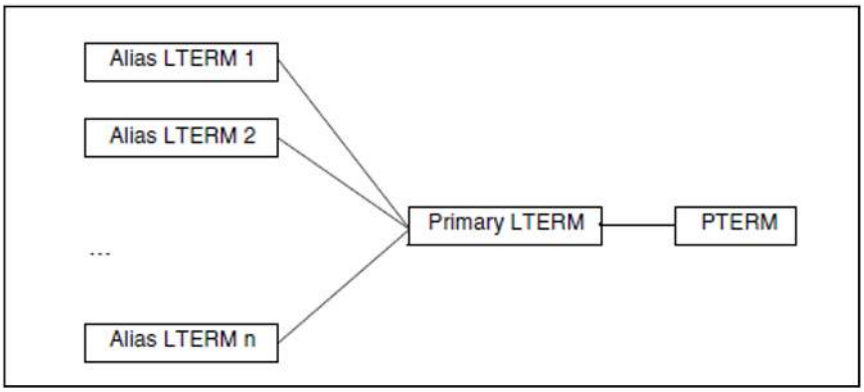


Figure 11: Example of a LTERM group

LTERM groups can also be used in conjunction with LTERM bundles. In this case the primary LTERM is the master LTERM of the LTERM bundle.

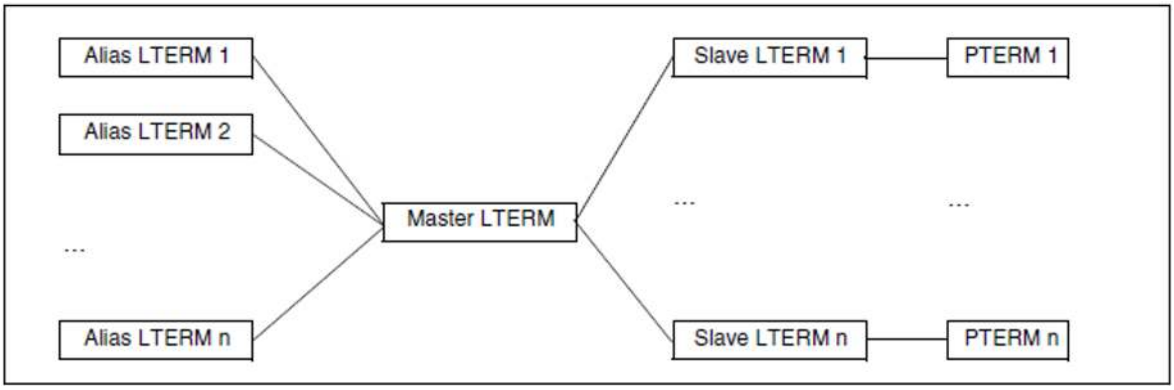


Figure 12: Example of a LTERM group in conjunction with a LTERM bundle

FPUT/DPUT calls

FPUT and DPUT calls sent by program units to an alias LTERM are processed as follows:

- In a LTERM group without a LTERM bundle:

  FPUT and DPUT calls sent to an alias LTERM are sent by openUTM via the PTERM assigned to the primary LTERM.

- In a LTERM group whose primary LTERM is the master LTERM of a LTERM bundle:

  If FPUT calls are sent to an alias LTERM in this kind of LTERM group, then all queued messages sent in the transaction to alias LTERMs in the group are assigned by openUTM to exactly one of the slave LTERMs at the end of the transaction.

  This procedure guarantees that the recipient receives the messages in the same order as they would be generated in a transaction for an LTERM group.

Program units can also send FPUT and DPUT calls directly to the primary LTERM.

Information displayed in the KB header

If the primary LTERM of a LTERM group is not the master LTERM of a LTERM bundle, then messages can also be received via the primary LTERM. In services started for received messages, openUTM always displays the name of the LTERM or LPAP from which the message was received in the KB header. The following also applies to LTERM groups:
In services started for messages received via the primary LTERM, the name of the primary LTERM is displayed in the KB header and **not** the name of an alias LTERM.

With the help of the KDCS call INIT PU you can obtain information on whether the LTERM in the KB header is the primary LTERM of a LTERM group (see openUTM manual „Programming Applications with KDCS”).

LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
In addition to the properties forLTERM partners already listed (see "Connecting clients via LTERM partners"), the following operands must also be specified for a LTERM group:

- GROUP=

  Specifies the corresponding primary LTERM in the definition of an alias LTERM. The primary LTERM specified here must have been generated in a preceding LTERM statement:

  ```
  LTERM primary , ...
  PTERM primary , LTERM= primary , PTYPE=APPLI | SOCKET, ...
  LTERM alias1 , GROUP= primary , ...
  LTERM alias2 , GROUP= primary , ...
  ```

> All LTERM parameters of the alias LTERMs except for *ltermname*, USER, and STATUS must match the same parameters of the primary LTERM. Otherwise they will be overwritten by KDCDEF using the data specified in the primary LTERM. No message is output in this case.
>
> Only the generation parameters of the primary LTERM are evaluated for a FPUT or DPUT call.

PTERM statement in section "PTERM - define the properties of a client/printer and assign an LTERM partner"
In addition to specifying the properties for physical clients already listed (see "Connecting clients via LTERM partners"), the following operands must also be specified for the PTERM assigned to the primary LTERM of a LTERM group:

- PTYPE=APPLI | SOCKET

  The PTERM in a LTERM group must be generated with PTYPE=APPLI or PTYPE=SOCKET.
- USAGE=D (BS2000 systems only)

  The PTERM in a LTERM group must be generated with USAGE=D.

4.1.5 Connecting OpenCPIC clients

OpenCPIC clients are treated as OSI TP partners. For this reason, this section only describes the client-specific features of OSI TP generation.

Generation

An OpenCPIC client is generated in a similar way to a server/server link, see "Distributed processing via the OSI TP protocol". The only difference is that LTAC statements are not required if it is just a client.

4.1.6 Defining the client sign-on procedure

This section describes the interface between generation and the sign-on procedure for clients if the application is generated with user IDs. The sign-on procedure is made up of the following two stages: **establishing the connection** and **signing on**.

The connection is established using the application names as specified in the operand BCAMAPPL= or, for OpenCPIC, specified in the operand LOCAL-ACCESS-POINT=.

Signing on to a UTM application

Signing on to a UTM application is carried out using a user ID. The following stages are required, regardless of whether the default sign-on service is used or another sign-on service:

- When using terminals, the terminal user must prove their authorization once a connection has been established. To do this the user must enter at least one user ID. This user ID must be generated in a USER statement. This is also called the **real user ID**.
- TS applications, UPIC clients and HTTP clients are signed on after connection using a so-called **connection user ID**. This is a user ID which is
  - either a user ID implicitly generated by openUTM using the LTERM name if no user ID is specified in the operand USER= of the LTERM statement,
  - or an explicit connection user ID, if a user ID is specified in the operand USER= which is generated with a USER statement, see "Automatic sign-on under a specific user ID". This user ID cannot be used as a real user ID.
- OpenCPIC clients are signed on under their **association names** once the association has been established. The association names are formed using the names specified in the operand ASSOCIATION-NAMES= and a sequential number, for example, ASSOC03, see "OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP"

UPIC clients, HTTP clients, TS applications and OpenCPIC clients which are signed on using a connection user ID or using their association name can then subsequently sign on using a real user ID.

The execution of the sign-on service can be defined by the generation, for example, using automatic connection establishment, automatic sign-on under a specific user ID, separate sign-on service or by permitting multiple sign-ons.

> **i** Detailed information about the sign-on service can be found in the relevant section of the openUTM manual “Using UTM Applications”. The individual steps required for a client to sign on to a UTM application are described there.

4.1.6.1 Establishing an automatic connection

Certain clients can be generated in such a way that openUTM attempts to establish a connection to the client as soon as the application is started. This is possible when using:

- OpenCPIC clients,
- individually generated terminals and TS applications on BS2000 systems
- individually generated TS applications on Unix, Linux and Windows systems.

An automatic connection can be established as follows:

- For TS applications and terminals:

  `PTERM ... ,CONNECT=YES`
- For OpenCPIC clients:

  `OSI-LPAP ... ,CONNECT=`*`n`* `(`*`n`*`>0)`

4.1.6.2 Automatic sign-on under a specific user ID

You can explicitly assign all clients defined using LTERM/PTERM a user ID under which this client automatically signs on once a connection is established. If you do so, the authorizations that apply to that client are the ones assigned to the specified user ID, see "Generating security functions". To do this you will need the following generation statements:

`LTERM ... USER=`*`username`*
`USER` *`username`* `...`

Terminals are then always signed on under this user ID. For TS applications and UPIC clients this user ID is a connection user ID and can still be replaced by a real user ID, e.g. in a sign-on service, see below.

However, after signing off via KDCOFF BUT a user may sign-on again with another user ID.

4.1.6.3 Generating sign-on services for clients

It is possible to program special sign-on services for terminals, UPIC clients and TS applications. A sign-on service is linked to an application name. This means that you can assign a sign-on service to any application name. Application names are defined using MAX APPLINAME= or in a BCAMAPPL statement.

If a client signs on using a specific application name, then the sign-on service assigned to this application name is started. The application name under which the client has to sign on is specified in PTERM/TPOOL in the operand BCAMAPPL.

Sign-on services are generated as follows:

- The sign-on service for the default application name (as defined in MAX ... APPLINAME) is generated using:

  `TAC KDCSGNTC, PROGRAM=`*signon-prog1*
  `PROGRAM` *signon-prog1* `...`

  signon-prog1 is the name of the program unit that is initially run in the sign-on service.

  If a default application name is generated for a sign-on service, this is then taken as the default value for all application names generated using BCAMAPPL.

- The sign-on service for an application name defined using BCAMAPPL is generated using:

  `BCAMAPPL` *appliname2...,*`SIGNON=`*signon-tac*
  `TAC` *signon-tac*`, PROGRAM=`*signon-prog2*
  `PROGRAM` *signon-prog2*

  signon-prog2 is the name of the program unit that is initially run in the sign-on service.

- If sign-on services are also to be run for UPIC clients, you must specify the following in the SIGNON statement:

  `SIGNON ... UPIC=YES`

  If this setting is not made, no sign-on service is started for UPIC clients, not even if an appropriate sign-on service has been generated for the application name.

More information about the programming can be found in the openUTM manual „Programming Applications with KDCS”.

4.1.6.4 Multiple sign-ons

When using user IDs, it is usually only possible for a single client to sign on to an application at any given time, a second attempt to sign on under the same user ID is thus rejected.

If you want to enable multiple sign-ons for user IDs you must generate the following:

```
SIGNON ... MULTI-SIGNON=YES
```

This means that it is possible, at any given time, for several clients to sign on to an application under a single real user ID without a restartable service context (USER *username*... RESTART=NO), but only **one** of these clients may be a terminal client.

OpenCPIC clients that have selected the functional unit "Commit", may perform a multiple sign-on under any real user ID.

4.1.7 Specifying maximum waiting times for dialog prompting

In the KDCDEF control statement MAX, you can specify the maximum waiting times for dialog prompting using the operand TERMWAIT= and PGWTIME= as well as the IDLETIME= operand of the PTERM statement.

- The operand PGWTTIME= is used to set the maximum permitted length of the interval between the output of a dialog message to the client after a blocking call (for example, a PGWT call) and the subsequent dialog input. If no input is made during this period of time, openUTM is forced to interrupt the service.

  After a blocking call, the task / work process remains in a synchronous wait state until the next dialog entry and cannot handle any other jobs until this time.
- The operand TERMWAIT= is used to set the maximum permitted length of the interval between the dialog output at a terminal after a PEND KP and the subsequent dialog input. If no input is made during this period of time, openUTM is forced to interrupt the service.

  After a PEND KP, any resources in the application maybe locked (e.g. database tables) and other users have to synchronously wait for these locks to be released.
- The operand IDLETIME= is used to limit the waiting time after PEND RE and PEND FI/ER/FR, or in other words, after the end of a transaction.
  Monitoring the waiting time after the end of a transaction is used for data protection purposes. Should a user forget to sign off after completing work with the application, this function allows you to reduce the risk of unauthorized persons may get access to the client and can work with the UTM application **without** being forced to sign on.

> ¡ The attention of the user must be drawn towards these relationships. The users must be shown the critical points of the dialog interaction so that they are aware of the effect an input delay may have on the performance of the application as a whole.

4.1.8 Generating security functions

The security functions are made up of the following components:

- System access control:
  System access control is defined in the USER statement, see below.
- Administration authorization:
  Administration authorization is assigned in the USER statement or the OSI-LPAP statement, see "Assigning administration authorizations".
- Data access control:
  Data access control is specified using the operands KSET, USER-KSET or ASS-KSET of the USER, LTERM, TPOOL or OSI-LPAP statement. Data access protection must be defined within the framework of a lock/key code concept or of the access list concept and is described in detail in section "Data access control". Data access control for OpenCPIC clients is generated in the same way as described in section "Protection measures for job-receiving services" (Data access control with distributed processing).
- Encryption:
  openUTM is supporting message encryption for the communication with specific clients. The encryption level is specified in the operand ENCRYPTION-LEVEL of the PTERM, TPOOL or TAC statement. A detailed description of message encryption can be found in section “Message encryption on connections to clients”.

4.1.8.1 Defining system access control

System access control is only relevant for real user IDs and is generated in the USER statement.

You define system access control by assigning a password to each user ID and by specifying a minimum length, a certain level of complexity, and a minimum and a maximum validity period for the password.

USER *userid-name*,PASS=*password*,PROTECT-PW=*complexity-level*

On signing on, the user must pass the specified authorization data configured for *userid-name* to openUTM.

On BS2000 systems, a magnetic strip card can also be configured as an access requirement for a user ID.

In addition, on BS2000 systems an access check using Kerberos can be generated as an alternative to a password and/or magnetic strip card.

Generating system access control using Kerberos (BS2000 systems)

The following generation statements are of significance for generating access control using the distributed authentication service Kerberos:

- LTERM KERBEROS-DIALOG=

  If you specify LTERM KERBEROS-DIALOG=YES, a Kerberos dialog is carried out when a connection is established for terminals that support Kerberos and that connect to the application directly via this LTERM partner (not via OMNIS) (see "LTERM - define an LTERM partner for a client or printer").
- TPOOL KERBEROS-DIALOG=

  If you specify TPOOL KERBEROS-DIALOG=YES, a Kerberos dialog is carried out when a connection is established for terminals that support Kerberos and that connect to the application directly via this terminal pool (not via OMNIS) (see "TPOOL - define an LTERM pool").
- USER PRINCIPAL=

  When you specify USER PRINCIPAL=characterstring, the user is authenticated via Kerberos with the help of this string (see "USER - define a user ID").

openUTM stores the Kerberos information in the length resulting from the maximum lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH (see "MAX - define UTM application parameters"). If the Kerberos information is longer, it is truncated to this length and stored.

4.1.8.2 Assigning administration authorizations

- You can specify the administration authorizations for a user ID using the USER statement:

  `USER` *userid-name*`,PERMIT=ADMIN`

  On BS2000 systems you can also assign the UTM-SAT administration authorizations for a user ID (PERMIT operand) and specify the type and range of the SAT logging (SATSEL operand):

  `USER` *userid-name*`,PERMIT=SATADM,SATSEL=...`

- You can assign administration authorization to an OSI TP-Partner, e.g. an OpenCPIC client in OSI-LPAP:

  `OSI-LPAP ... PERMIT=ADMIN`

  On BS2000 systems you can also assign the UTM-SAT administration authorizations for a client:

  OSI-LPAP ... PERMIT=SATADM or PERMIT=(ADMIN,SATADM)

  If the OSI TP-Partner signs on under a real user ID, then the data access rights that are generated for that user ID are valid and not the data access rights of the OSI-LPAP.

4.1.9 Generating restart functionality

The restart function for a client is linked to the user ID that the client has used to sign on to the UTM application.

Restart function for real user IDs

The restart function for real user IDs is specified in the RESTART operand of the USER statement.

`USER` *userid-name*`...RESTART=YES | NO`

If this is generated as RESTART=YES then the type of client and any generated sign-on services will also play a role in service restart:

- If a sign-on service has been generated for a client that signs on using this user ID, then this service will control whether a service restart is performed or whether an open service is terminated abnormally, see the description of the sign-on service in the openUTM manual „Programming Applications with KDCS”.
- If a terminal or TS application does not sign on via a sign-on service, then openUTM always initiates a service restart.
- If a UPIC client does not sign on via a sign-on service then the UPIC client must explicitly initiate a service restart, otherwise an open service is terminated abnormally see the manual „openUTM-Client for the UPIC Carrier System”.
- When using OpenCPIC clients, restart is only possible with cooperative processing (functional unit not equal to "Commit"). The OpenCPIC client must explicitly initiate a service restart, otherwise an open service is terminated abnormally, see manual “openUTM-Client for the OpenCPIC Carrier System”.

Restart function for connection user IDs

If individually generated TS applications sign on via implicit (created by openUTM) connection user IDs, then the restart function is controlled by the RESTART operand in the LTERM statement:

`LTERM` *ltermname* `... RESTART=YES | NO`

This parameter of the LTERM statement is irrelevant for the service restart if the TS application is signed on via an explicitly generated connection user ID or a real user ID.

> No service restart is possible for UPIC clients that do not sign on under a genuine user ID.
> Explicitly generated connection user IDs to UPIC clients are generated in any case and without any message with RESTART=NO.

4.1.10 USP headers for output messages to USP socket applications

In order that a UTM application is able to communicate with the TS application via the socket interface, a UTM socket protocol (USP) may be used on top of TCP/IP. openUTM uses this protocol to convert a bytestream received via the socket interface into a message. The partner application must issue the protocol and prefix it with the input message as the protocol header. openUTM does not usually create a protocol for output messages.

It is possible to set each generation option so that openUTM also prefixes a protocol header for output messages. This is specified in the PTERM or TPOOL statement using the operand USP-HDR=:

- USP-HDR=ALL ensures that openUTM prefixes all output messages of this connection (dialog, or asynchronous message, K message) with a protocol header.
- With USP-HDR=MSG the protocol header is prefixed for K messages only.
- USP-HDR=NO means that no protocol header is prefixed for output messages.

The structure of the protocol header is described in the openUTM manual „Programming Applications with KDCS”.

4.1.11 Providing address information

For TS applications, HTTP clients, UPIC clients and OpenCPIC clients, address information is required to establish a connection. This information is stored in the UTM generation. For socket applications, there is no difference between BS2000 systems and Unix, Linux or Windows systems. For other clients, the characteristics of the transport system take effect. This information is therefore divided into separate paragraphs.

The address information for OpenCPIC clients is specified in the same way as for OSI TP partners in generell, see chapter "Generation of distributed processing based on OSI TP". The information supplied must match the configuration of the OpenCPIC client.

Providing the address information for HTTP clients and TS applications of type SOCKET

When communicating with HTTP clients or TS applications via TCP/IP the socket interface is used directly.

The address information required for communication is provided in the UTM generation.

KDCDEF generation

The address information is stored in the operands LISTENER-PORT=, T-PROT= and PRONAM= of the BCAMAPPL and TPOOL/PTERM statements.

- BCAMAPPL statement

  You must always specify a BCAMAPPL statement for socket applications. The following applies:

  - In LISTENER-PORT= you must always enter a port number under which the UTM application waits for the requests of the socket application. The port number must correspond to the settings of the communication partner.
  - In T-PROT= you must enter SOCKET.
  - Unix, Linux and Windows systems:
    LISTENER-ID= assigns the connection an optional listener ID. You can use the listener ID to distribute the management of network connections to different network processes. The values for the LISTENER-ID of non-socket connections and socket connections may be assigned independently of each other.

- PTERM statement

  If you generate a socket partner individually, you will need to specify the following operands and parameters:

  - If the socket application is to establish the connection, the PTERM name must have the format PRT*nnnnn*, where *nnnnn* is the port number from which the socket application establishes the connection. The name may be supplemented by leading zeros.
  - In BCAMAPPL= enter the application name as defined in the BCAMAPPL statement.
  - In PRONAM= you must specify the TCP/IP host name.

    For Unix, Linux and Windows systems, please consider also section "Specifying computer names in KDCDEF generation".
  - In the LISTENER-PORT= operand, you must specify the port number at which the socket partner is waiting for connection establishment requests.

- TPOOL statement

  If you want to connect a socket partner using a LTERM pool, then:

  - In PRONAM= you must specify the TCP/IP host name or the mapped name of a SUBNET statement.
    If you specify the mapped name of a SUBNET statement for PRONAM, clients from any computers from this subnet can sign on, provided they correspond to the type specified in PTYPE=.

If you enter PRONAM=*ANY, then clients from any computer can sign on assuming they are of the same type as specified in PTYPE= .

For Unix, Linux and Windows systems, please consider also section "Specifying computer names in KDCDEF generation".

- In BCAMAPPL= enter the application name as defined in the BCAMAPPL statement.

Example

The following example uses the port number 10100. The socket application is linked via the LTERM pool and runs on the computer PCSOCK01.

```
BCAMAPPL BSPSOCK -
       ,LISTENER-PORT=10100 -
       ,T-PROT=SOCKET -
       ,...

TPOOL  ...
       ,PTYPE=SOCKET -
       ,BCAMAPPL=BSPSOCK -
       ,PRONAM=PCSOCK01 -
       ,...
```

4.1.11.1 Providing address information for clients of type UPIC and APPLI on BS2000 systems

For RFC1006 (or ISO) connections you must always generate a separate application name. This can be used for all RFC1006 (or ISO) connections regardless of client type.

KDCDEF generation

- BCAMAPPL statement

  In T-PROT= enter either ISO or RFC1006 (these two values are treated identically on BS2000 systems).

- PTERM statement

  If you want to generate the clients individually, specify the following operands and parameters:

  - Under PTERM name, you must enter the name that was defined for this client when the network was generated.
  - In BCAMAPPL= enter the application name defined in the BCAMAPPL statement.
  - In PRONAM= enter the name of the computer on which the client is running. This name is specified when the network is generated.
  - In PTYPE= you must enter either UPIC-R or APPLI.

    If you want the UTM application to actively establish the connection to clients of the APPLI type, in the LISTENER-PORT parameter you must specify the port number on which the partner application listens for connection requests.

- TPOOL statement

  If you want to link clients via the LTERM pool, then:

  - In BCAMAPPL= enter the application name that is defined in the BCAMAPPL statement.
  - In PRONAM= you can enter the name of the computer on which the clients are running.

    If you specify PRONAM=*ANY the clients can sign on from any computer as long as it is of the type specified in PTYPE=.
  - In PTYPE= you must enter either UPIC-R or APPLI.

4.1.11.2 Providing address information for clients of type UPIC and APPLI on Unix, Linux and Windows systems

On Unix, Linux and Windows systems it is possible to write your own BCAMAPPL statement for an application name that is generated using MAX APPLINAME=. This statement can then be used to specify all the parameters as required by the application.

To link clients via RFC1006, see section "Providing address information for the CMX transport system (Unix, Linux and Windows systems)". The KDCDEF generation must contain all the necessary address information.

KDCDEF generation for RFC1006

- BCAMAPPL statement
  - *appliname*: You can select any application name, but the name must be unique within the network, as KDCDEF uses it to create a T-selector.
  - If OPTION CHECK-RFC1006=YES then a port number must be specified for LISTENER-PORT.
    In all other cases, the default value is 0 (no port number).
  - In T-PROT you must enter RFC1006.
  - In TSEL-FORMAT= enter the format indicator for the name that you have defined as the *appliname* (see above). It is recommended that you always make an entry for the operand TSEL-FORMAT=.
- PTERM statement

  If you want to generate the client individually via a PTERM statement, enter the following:
  - As PTERM name use the T-selector of the client. The client on the client computer must be entered with this T-selector as the local application.
  - In BCAMAPPL= enter the application name as defined above.
  - In LISTENER-PORT= enter the port number which the client is reached as output port on the client computer.
  - In PRONAM= you must enter the TCP/IP host name, see "Specifying computer names in KDCDEF generation ".
  - In PTYPE= you must specify wither UPIC-R or APPLI.
- TPOOL statement

  If you want to connect the client via an LTERM pool, enter the following:
  - In BCAMAPPL= enter the application name as defined above.
  - In PRONAM= you can enter the name of the computer on which the client is running. This must be the TCP/IP host name, see "Specifying computer names in KDCDEF generation".

    If you enter PRONAM=*ANY the clients can sign on from any computer as long as it is of the same type specified in PTYPE=.

    If you specify the mapped name of a SUBNET statement for PRONAM, clients from any computers from this subnet can sign on, provided they correspond to the type specified in PTYE=.
  - In PTYPE= you must enter either UPIC-R or APPLI.

Example

The following example uses the port number `10030` locally and the remote application has the port number `12030`. The UPIC client runs on a computer called `PCUPR`.

```
BCAMAPPL BSPUPR -
       ,LISTENER-PORT=10030 -
       ,T-PROT=RFC1006 -
       ,T-SEL-FORMAT=T -
       ,...

PTERM UPRPART -
       ,PTYPE=UPIC-R -
       ,BCAMAPPL=BSPUPR -
       ,PRONAM=PCUPR -
       ,LISTENER-PORT=12030 -
       ,T-PROT=RFC1006 -
       ,T-SEL-FORMAT=T -
       ,...
```

The statements for a TS application are created analogue, except that you must specify PTYPE=APPLI in PTERM.

4.1.11.3 Additional information for LTERM pools on Unix, Linux and Windows systems

An LTERM pool can be used by any partner application on a specific computer to establish a connection to the UTM application, if the partner application is of the appropriate type (PTYPE). If PRONAM=*ANY is generated in the TPOOL statement, then partner applications of the generated type can connect to the UTM application from any computer.

If you specify the mapped_name of a SUBNET statement for PRONAM, clients from any computers from this subnet can sign on, provided they correspond to the type specified in PTYE=.

The TPOOL statement does not specify a name (station name) for this communication partner. The UTM application determines this name from the transport address of the partner application that the partner application has supplied when the connection is established.

The following procedure is used:

- If a partner application is communicating with the UTM application, an attempt is made to find out the computer name using the local Name Service.
- If no name can be located for the computer the network process assigns it the name *ANY.
- Then an attempt is made to obtain the T-selector from the transport address. If a T-selector is found then it is used as the station name.
- If no T-selector can be found, then the station name 'NET*nnnnn*' is used for the partner application. *nnnnn* stands for a number between 00000 and 99999 and is automatically incremented by openUTM.

It often makes sense to communicate with LTERM pools via TCP/IP connections, if LTERM pool is generated with processor names (TPOOL ...,PRONAM=).

> **!** **Unix, Linux and Windows systems**
>
> Please note the maximum number of connections that can be established at a time via one transport system end point. For details see **BCAMAPPL statement** in section "BCAMAPPL - define additional application names".

4.1.12 Examples for the generation of a client/server combination

The following examples show how to connect a UPIC client which is running on a Windows PC to a UTM application on BS2000 and on a Unix or Linux system.

Example 1: Connecting a UPIC client to openUTM on BS2000 systems

The UTM server application is located on a host with the name BS2HOST1, the client program is running on a PC with the computer name PCCLT002. The transport connection is to be established via TCP/IP (address format RFC1006).

- UTM generation under the BS2000 system

```
*** Define BCAM application name for the UTM server application:***
BCAMAPPL SERVER, T-PROT=RFC1006

*** Generate client:***
PTERM UPICPT, PTYPE=UPIC-R, LTERM=UPICLT, BCAMAPPL=SERVER, -
      PRONAM=PCCLT002
LTERM UPICLT

*** Define TAC for the client:***
TAC TAC1, PROGRAM=SERVICE
```

  The statement LTERM UPICLT means that openUTM implicitly uses a connection user ID called UPICLT.

- Entries in the side information file (upicfile) of the UTM client

```
* UTM application under the BS2000 system
SDsamplaw SERVER.BS2HOST1 TAC1
* or, if you require automatic conversion of user data
* from ASCII to EBCDIC and vice versa
HDsamplaw SERVER.BS2HOST1 TAC1
```

- Specification in the client program

```
Enable_UTM-UPIC "UPICPT"
Initialize_Conversation "samplaw"
```

Example 2: Connecting a UPIC client to openUTM on Unix, Linux or Windows systems

This example describes the TCP/IP RFC1006 connection of a UPIC client to a UTM application on a Unix, Linux or Windows system. The example shows the coordination of the generation for both communication partners.

The UTM application is running on a computer with the name `UXHOST01`. The client is located on a Windows system for which the name `PCCLT001` has been generated in the KDCDEF. The UTM application receives the local port number 1230.

- Generating the UTM server on the Unix, Linux or Windows system

```
BCAMAPPL UTMUPICR,LISTENER-PORT=1230,T-PROT=RFC1006,TSEL-FORMAT=T

PTERM UPICPT,PTYPE=UPIC-R,LTERM=UPICLT,BCAMAPPL=UTMUPICR,PRONAM=PCCLT001, \
            T-PROT=RFC1006,TSEL-FORMAT=T
LTERM UPICLT
```

  The statement LTERM UPICLT means that openUTM implicitly uses a connection user ID called UPICLT.

- Entries in the side information file of the client computer

```
* Local application
LNUPICTTY UPICPT PORT=1240

* UTM application on Unix/Linux/Windows system with port 1230,
* TCP/IP host name=UXHOST01
SDsampladm UTMUPICR.UXHOST01 TAC1 PORT=1230
```

- Specification in the client program

```
Enable_UTM_UPIC "UPICPT"
Initialize_Conversation "sampladm"
```

Example 3: Connecting an OpenCPIC client

An OpenCPIC client is running on a Unix or Linux system with the host name UNIXPRO1. The client connects itself via RFC1006 to a UTM application on BS2000 system and to a UTM application on Unix, Linux or Windows system. The following is to apply:

- In the UTM application on the BS2000 system, the client calls the transaction code TRAVEL02 and in the UTM application on Unix, Linux or Windows system it calls the transaction code STATIST1.
- It is to be possible to have up to 10 parallel connections to the BS2000 system, and up to 2 parallel logical connections to Unix, Linux or Windows system.
- The UTM application on the BS2000 system uses the local port number 102. The UTM application on Unix, Linux or Windows system uses the local port number 12000.
- The OpenCPIC client uses the local port number 13000.

- UTM generation on BS2000 system

```
UTMD APT = (2, 7, 16, 2)

ACCESS-POINT SERVER,
   T-PROT = RFC1006,
   P-SEL = *NONE,
   S-SEL = *NONE,
   T-SEL = C'UTMSERV1',
   AEQ = 1

OSI-CON CONNECTB,
   LOCAL-ACCESS-POINT = SERVER,
   P-SEL = *NONE,
   S-SEL = *NONE,
   T-SEL = C'CPICCLT1',
   N-SEL = C'UNIXPRO1',
   LISTENER-PORT = 13000,
   OSI-LPAP = OSILPAPB

OSI-LPAP OSILPAPB,
   APT = (2, 7, 16, 4),
   APPLICATION-CONTEXT = UDTSEC,
   AEQ = 1,
   ASS-NAMES=CPIC,
   ASSOCIATIONS=10,
   CONTWIN=0
TAC TRAVEL02 ...
```

- UTM generation on Unix, Linux or Windows system

```
UTMD APT = (2, 7, 16, 3)

ACCESS-POINT STATSERV,
   T-PROT = RFC1006,
   P-SEL = *NONE,
   S-SEL = *NONE,
   T-SEL = C'UTMSERV2',
   LISTENER-PORT = 12000,
   T-PROT = RFC1006,
   T-SEL-FORMAT = T,
   AEQ = 1

OSI-CON CONNECTX,
   LOCAL-ACCESS-POINT = STATSERV,
   P-SEL = *NONE,
   S-SEL = *NONE,
   T-SEL = C'CPICCLT1',
   N-SEL = C'UNIXPRO1',
   LISTENER-PORT = 13000,
   T-PROT = RFC1006,
   T-SEL-FORMAT = T,
   OSI-LPAP = OSILPAPX

OSI-LPAP OSILPAPX,
   APT = (2, 7, 16, 4),
   APPLICATION-CONTEXT = UDTSEC,
   AEQ = 1,
   ASS-NAMES=CPIC,
   ASSOCIATIONS=2,
   CONTWIN=0

TAC STATIST1 ...
```

- OpenCPIC generation

```
*** Entry for the local application
LOCAPPL OPENCPIC,
   APT = (2, 7, 16, 4),
   AEQ = 1

*** Connection to UTM application on BS2000 system
PARTAPPL UTMSBS20,
   APT = (2, 7, 16, 2),
   APPLICATION-CONTEXT = utm-secu,
   AEQ = 1,
   ASSOCIATIONS = 10,
   CONTWIN = (10,10),
   CONNECT = 10***

*** TAC in the UTM application on BS2000 system
SYMDEST TRAVEL,
   PARTNER-APPL = UTMSBS20,
   PARTNR-APRO = TRAVEL02

*** Connection to UTM application on Unix, Linux or Windows system
PARTAPPL UTMSUNIX,
   APT = (2, 7, 16, 3),
   APPLICATION-CONTEXT = utm-secu,
   AEQ = 1,
   ASSOCIATIONS = 2,
   CONTWIN = (2,2),
   CONNECT = 2

*** TAC in the UTM  application on Unix, Linux or Windows system
SYMDEST STATIST,
   PARTNER-APPL = UTMSUNIX,
   PARTNR-APRO = STATIST1
```

- TNS entries in the OpenCPIC client computer (tnsxfrm format)

```
OPENCPIC\
   PSEL V''
   SSEL V''
   TSEL RFC1006 T'CPICCLT1'
   TSEL LANINET A'13000'

UTMSBS20\
   PSEL V''
   SSEL V''
   TA RFC1006  ip-address-bs2  PORT 102 T'UTMSERV1'

UTMSUNIX\
   PSEL V''
   SSEL V''
   TA RFC1006  ip-address-unix  PORT 12000 T'UTMSERV2'
```

4.2 Generating printers (on BS2000, Unix and Linux systems)

> Printers cannot be generated in UTM applications on Windows systems.

Printers that are to be used by a UTM application are connected via LTERM partners that are configured with the logical properties for printers. LTERM partners for printers are defined in the LTERM statement. Printers **cannot** be connected via LTERM pools. Physical printers are defined with the PTERM statement, which is also where the assignment is made to the LTERM partner.

The connection setup by openUTM can be defined either with PTERM...,CONNECT=YES or with LTERM...,PLEV=. The connection can also be established using administration functions. See also the openUTM manual “Administering Applications”.

LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
The most important properties for LTERM partners via which printers can connect to an application are defined with the following operands:

- ltermname

  Name of the LTERM via which the printer is connected to the UTM application.
- CTERM=

  Defines the printer control LTERM so that the user can administer printers, print jobs, and the print jobs in the message queue of the LTERM partner.
- PLEV=

  Number of printer messages for which openUTM attempts to establish a connection to the printer assigned to this LTERM partner.
  - If PLEV=1, a connection is established for each print job.
  - If PLEV=*n*, the connection is established for the *n*th print job (n=1 to 32767).
  - If PLEV=0, the connection setup is not initiated by pending print jobs, rather is initiated explicitly by the administrator using the command KDCLTERM...ACT=CON or KDCPTERM.

    The connection is shut down again as soon as there are no further messages for this printer.

    PLEV= makes it easier for the user to use printers from various UTM applications (printer sharing). In this case, the connection between a UTM application and the printer is only kept open while the print job is being transmitted, in order to allow other applications to establish a connection. If there are unprocessed print messages for a printer in the message queue of the LTERM partner when the UTM application terminates, these print messages are retained until the next application start. Before terminating the application, the administrator can initiate the processing of the outstanding print messages using the command KDCAPPL SPOOLOUT=ON.
- QAMSG=

  Messages to printers can be buffered in the message queue of the LTERM partner, even if the printer is not connected to the application.
- USAGE=O

  USAGE=O defines a printer as a communication partner which can connect to an application via the LTERM partner. Messages can only be sent from the application to the printer.

Each printer must be described in the configuration, i.e. for each printer, a PTERM statement is written with the physical properties of the printer and the assignment is made to an LTERM partner. The assignment between the printer and LTERM partner is static, i.e. the assignment applies until it is canceled using administration commands. You can assign another printer (of the same type) to an LTERM partner during operation, e.g. in the event of a printer fault.

PTERM statement in section "PTERM - define the properties of a client/printer and assign an LTERM partner"
The most important properties for printers are defined with the following operands:

- ptermname

  Name of the printer

  BS2000 systems:
  The BCAM name or the RSO name of the printer must be specified.

  Unix and Linux systems:
  The name of a printer group of the spool system must be specified. A printer group used by UTM applications should comprise only **one** printer. This is the only way to ensure that all parts of a message are output to the same printer if a message comprises message segments (see also the information on printer pools).

- CID=

  The printer is assigned a printer ID *printer_id* via which the printer can be identified by a printer control LTERM. The printer control LTERM attaches to the LTERM partner to which the printer is assigned.

- CONNECT=

  Specifies whether or not openUTM automatically establishes a connection to the printer when the application starts. The printer is then explicitly occupied by the application until the next time the connection is cleared down, even if there are no print jobs.

- LTERM=

  Name of the LTERM partner assigned to the printer *ptermname* and via which the printer is connected to the UTM application.

- PTYPE=

  BS2000 systems:
  Printer type or *RSO.

  Unix and Linux systems:
  Printer type.
  To output the data, the printer process (*utmprint*) calls the *utmlp* script. The call also passes parameters to *utmlp* in addition to the data to be printed. By default, *utmlp* then passes the data to the lp command (see PTYPE=PRINTER on "PTERM - define the properties of a client/printer and assign an LTERM partner").

- USAGE=O (only BS2000 systems)

  The communication partner is a printer.

The maximum values and limits that are to apply throughout the application for printers are defined with the MAX statement.

MAX statement in section "MAX - define UTM application parameters"
The default and maximum values relevant throughout the application for printers are defined with the following operands:

- CONRTIME=

  Time in minutes after which openUTM makes cyclical attempts to reestablish a logical connection. openUTM attempts this for:

  - Printers to which openUTM establishes a connection as soon as the number of print jobs for this printer exceeds the generated threshold value (LTERM...,PLEV>0). When the connection is aborted, the number of print jobs must be greater than or equal to the threshold value if openUTM is to attempt to re-establish the connection.
  - Printers to which openUTM automatically establishes a connection (PTERM...,CONNECT=YES), provided that the connection was not terminated by the administration.

  If no connection is established when the application starts or if the connection is interrupted during operation, openUTM attempts to reestablish the connection at intervals of CONRTIME=.

- PGPOOL=

  A sufficiently large value must be specified for the PGPOOL operand so that the page pool can accommodate all print messages in the event of a high print volume.

- LOGACKWAIT=

  Maximum time in seconds that openUTM is to wait for an acknowledgment from the output devices. This acknowledgment is

  - for a printer, the logical print acknowledgment from the printer,
  - for an RSO printer, the acknowledgment from RSO,
  - with an FPUT call to another application, a transport acknowledgment.

4.2.1 Generating RSO printers (BS2000 systems)

Via the OLTP interface of RSO (remote spool output), openUTM obtains access to all printers that support RSO, i.e. including printers connected via LAN or PC. openUTM does not establish a transport connection to these printers, rather serves them via the OLTP interface, i.e. openUTM reserves the printer for RSO and transfers the print job to RSO.

4.2.1.1 Entries for the KDCDEF generation

To print to an RSO printer from openUTM, the desired printer is defined in the generation under its RSO name as an RSO printer in the PTERM statement. The printer must be defined and activated on the RSO side. This section only lists the RSO specific statements and operands, the other printer-specific parameters are described on "Generating printers (on BS2000, Unix and Linux systems)"

PTERM statement in section "PTERM - define the properties of a client/printer and assign an LTERM partner"
RSO printers served by openUTM via the OLTP interface are defined with the following operands:

- ptermname

  For an RSO printer, the name of the printer must be specified here as it was defined in RSO (logical RSO device name).
- PTYPE=

  PTYPE=*RSO is specified as the printer type. No particular printer type is specified for RSO printers. openUTM obtains the printer type in accordance with the RSO device information in the RSO call.
- PRONAM=

  For an RSO printer, *RSO must be specified as the processor name.

If RSO printers are defined, REMOTE-BUFFER-SIZE in the SPOOL parameter file must be >= 32700 (= value of MAX TRMSGLTH).

4.2.1.2 Entries for RSO and SPOOL

In order for openUTM to use the OLTP interface of RSO, RSO and the software products required by RSO must be installed. The RSO subsystem must be active. If you have specific questions, read the RSO manual:

Device definition

With the UTM tool SPSERVE, you open the SPOOL parameter file for the printer definition. The system administrator must configure the printer in RSO for UTM print jobs:

```
ADD-SPOOL-DEVICE...ADMINISTRATOR=*ADMINISTRATOR(...),
                   PROCESSING-CONTROL=*PAR(
                   DISCONNETION=*YES
                   RESET={*YES | *NO }
                   CONTROLLER-START=AT-PRINTER-START)
```

- If a new printer is configured, up to 8 RSO device managers can be entered with the parameter ADMINISTRATOR=*ADMINISTRATOR(...). An RSO device manager can modify a device with MODIFY-SPOOL-DEVICE or start a DEVICE with START-PRINTER-OUTPUT.
- It is advisable to work with the parameter DISCONNECT=*YES, because with SOCKETS the printer shuts down the connection when the time set on the printer has expired.
- The parameter CONTROLLER-START must be set to AT-PRINTER-START.
- If RESET=*YES, the settings of the printer menu are used. This also applies regardless of the device entry if openUTM is working with formats. If “logical” formats are used (see note at the end of the section) then openUTM behaves as if no formats are used.
  - If a format name is transferred by openUTM with an FPUT in the KCMF field, a RESET=*YES is sent by default to the printer by FHS before the message, so that the menu setting of the printer comes into effect before printing. In the printer menu you can set various fonts or CPI values, for example. In this case, RSO processes a message with format names as per the setting CONTROL=TRANSPARENT.
  - If no format name is transferred by openUTM with an FPUT in the KCMF field, RSO only sends a RESET to the printer before the message if RESET=*YES is entered in the device definition. If no RESET was sent to the printer, the applicable values are the printer menu values currently set on the printer, which may have been changed by a previous print job. RSO handles a printer message without format names as per the setting CONTROL=PHYSICAL.

The commands ADD-SPOOL-FORM for form entries and ADD-SPOOL-CHARACTER for character sets have no effect on UTM print jobs. If both UTM RSO print jobs and RSO SPOOLOUT are processed under the same logical RSO device name, forms and character sets are only relevant for the latter. As the only printer control character, RSO adds the RESET character string for UTM print jobs.

A UTM print job to an RSO printer is not placed in the SPOOL queue.

Sample device entry

Output of a device entry under which the printer is defined for RSO:

```
/show-spool-dev PGTP0041,inf=*all
DEVICE-NAME         : PGTP0041
DEVICE-TYPE         : 9021RP
------------------------------ DEVICE-ACCESS -------------------------------
DEVICE-ACCESS       : *TCP-ACCESS
ACCESS-TYPE         : *TACLAN
```

```
PROCESSOR-NAME      : *NONE
STATION-NAME        : *NONE
MNEMONIC-NAME       : *NONE
PROGRAM-NAME        : *NONE
INTERNET-ADDRESS    : PGTP0041
PORT-NAME           : 9100
LPD-PRINTER-NAME    : *NONE
FROM-PORT-NUMBER    : 0
TO-PORT-NUMBER      : 0
------------------------------ TWIN-DEVICE-DEF ------------------------------
SLAVE-MNEMONIC-NAME : *NONE
ESD-SIZE            : 0
---------------------------- DEVICE-INFORMATION ----------------------------
FORMS-OVERLAY-BUFFER: 32767
CHARACTER-SET-NUMBER: 64
ROTATION            : NO
DUPLEX-PROCESSING   : NO
FORMS-OVERLAY       : NO
RASTER-PATTERN-MEM  : *NONE
TRANSMISSION        : IGN
FONT-TYPE           : IGN
FACE-PROCESSING     : NO
MAXIMUM-INPUT-TRAY  : 1
MONJV               : NO
NOTIFICATION        : NO
ENCRYPTION          : NO
UNICODE             : NO
SUPP-FORMAT-NAME    :
   TEXT
   PLAIN-TEXT
------------------------------ ADMINISTRATOR ------------------------------
USER-IDENTIFICATION : *NONE
IDENTIFICATION      : OEC MW 135
TERMINAL            : PROCESSOR-NAME      :
                      STATION-NAME        :
---------------------------- SPOOLOUT-CONTROL ----------------------------
SHIFT               : 0
LINE-FEED-COMPRESS  : YES
BLANK-COMPRESSION   : YES
START-FORM-FEED     : YES
FORM-FEED           : *SINGLE-SHEET
                      DEFAULT-TRAY-NUMBER : 1
                      OUTPUT-TRAY-NUMBER  : 0
SKIP-TO-NEXT-PAGE   : BY-FORM-FEED
ESCAPE-VALUE        : NONE
--------------------------- PROCESSING-CONTROL ---------------------------
CONTROLLER-RESERVED : NO
FORM-NAME           : STD
DISCONNECTION       : YES
BUFFER-SIZE         : 1024
RESET               : YES
REPEAT-MESSAGE      : TYPE               : SYS
                      LIMIT              : NO
                      RETRY-TIME         : GLB
RESTART-ACTION      : LIMIT              : NO
                      RETRY-TIME         : GLB
SYNCHRONIZATION     : PRINTER
TIMEOUT-MAX         : 2
PAGE-EJECT-TIMEOUT  : NO
```

```
BAND-IDENTIFICATION : *NONE
LOAD                : NO
MODULO2             : NO
RECOVERY-RULES      : *SYSTEM
POLLING             : NO
PRINTER-PARAM-FILE  : *SYSTEM
RESOURCE-FILE-PREFIX: *SYSTEM
CONTROLLER-START    : AT-PRINTER-START
---------------------------- CHARACTER-SET-POS ----------------------------
POSITION-1          : N-U
POSITION-2          : N-U
POSITION-3          : N-U
POSITION-4          : N-U
POSITION-5          : N-U
POSITION-6          : N-U
POSITION-7          : N-U
POSITION-8          : N-U
POSITION-9          : N-U
POSITION-10         : N-U
POSITION-11         : N-U
POSITION-12         : N-U
POSITION-13         : N-U
POSITION-14         : N-U
POSITION-15         : N-U
POSITION-16         : N-U
------------------------------ MISCELLANEOUS ------------------------------
REDIRECTION-DEVICE  : *NONE
LANGUAGE-EXT-TYPE   : *SYSTEM
LINE-SIZE           : 150
CHARACTER-IMAGE     : *NONE
```

Defining the RSO buffer size

To be able to print out messages of any length, the RSO buffer must be greater than or equal to the maximum message length in openUTM. Since the maximum value for the UTM buffer size is 32 KB, the RSO buffer size in a session in which openUTM is running, must be adapted to this value:

```
/MODIFY-SPOOL-PARAMETER...SPOOLOUT-OPTIONS=*PAR(REMOTE-BUFFER-SIZE=32)
```

Any modification will come into effect in the next SPOOL session.

VTSU codes

When UTM messages containing VTSU codes are output to printers connected directly via BCAM, openUTM calls the VTSU program in order to convert VTSU codes into printer dependent escape sequences. If UTM messages are output to RSO printers, the conversion of the VTSU codes is carried out by RSO. An extensive adaptation to the known VTSU control characters was striven for in this case. Additional information can be found in the RSO manual.

4.2.1.3 Activating printers for openUTM

Each printer used in an RSO session must be started with START-PRINTER-OUTPUT. A START-PRINTER-OUTPUT statement can be contained in an ENTER file which is processed automatically when the RSO subsystem starts, or the system administrator or RSO device manager uses this statement to explicitly start the printer after the subsystem has started.

If you want to print to an RSO printer from openUTM, the printer must be released for openUTM:

```
/START-PRINTER-OUTPUT DEVICE-NAME=*RSO(NAME=devicename,
                                       ALLOWED-ACCESSES='UTM'))
                                        or
                                       ALLOWED-ACCESSES=('RSO','UTM'))
```

4.2.1.4 Querying printer information

Printer information in openUTM

The administration command KDCINF can be used in openUTM to query the current printer status. If required, the connection to the printer can be set up/shut down or locked using the administration commands KDCPTERM and KDCLTERM or via the program interface for administration. See also the openUTM manual “Administering Applications”.

Printer information in RSO

Using BS2000 information functions, users and RSO device managers can output the printer status of the RSO printers with the following command:

```
/SHOW-SYSTEM-STATUS INFORMATION=*REMOTE(DEVICE=NAME)
```

4.2.1.5 Releasing printers in the event of an error

If the automatic repetition of the print job by openUTM and RSO was not successful in the event of an error, the RSO device manager can temporarily release the printer with the command

```
/STOP-PRINTER-OUTPUT DEVICE-NAME=RSO-PRINTER(NAME=rso-printer,STOP=IMMEDIATE)
```

so that the printer can then be reserved again for openUTM with the START-PRINTER-OUTPUT command.

The parameter TRACE=YES can be specified under $TSOS for diagnostic purposes. In this case, a trace is generated and is stored under
$SYSSPOOL.SYSTRC.RSO.*devicename*.*date*.*time*:

```
/START-PRINTER-OUTPUT DEVICE-NAME=*RSO(NAME=devicename,
                                       TRACE=YES,
                                       ALLOWED-ACCESSES=(‘RSO‘,‘UTM‘))
```

See also the "RSO" manuals.

4.2.2 Generating printer pools

A printer pool is made up of several printers (=printer groups on Unix and Linux systems) assigned to one LTERM partner. A PTERM statement with the same *ltermname* for PTERM...,LTERM= is written for each printer in the pool. openUTM distributes the print output as evenly as possible to the printers in the pool. Messages that are made up of message segments are always output in full by openUTM to **one** printer or printer group (on Unix and Linux systems) in the pool.

4.2.3 Bypass mode (BS2000 systems)

With a locally connected printer, the term bypass mode is also used instead of spool mode. Bypass mode is possible if the terminal can conduct dialog independently of the print output. Bypass mode must only be implemented for terminal types 975x and 9763 or a corresponding emulation (see the manual "MT9750, 9750 Emulation on Windows")

4.2.4 Generating printer control LTERMs

In the generation you can define printer control LTERMs so that users themselves can administer the default printers and print job queues even without administration authorization, e.g. delete the current print job.

BS2000 systems

Each printer is assigned an LTERM partner, which is configured for an output medium (LTERM...,USAGE=O). All output jobs for this printer are “sent” by openUTM to the message queue of the associated LTERM partner, which thus becomes the print job queue. It is also possible to assign several printers to an LTERM partner (printer pool). In this case, all printers work with this print job queue.

A print control LTERM is an LTERM partner which is configured as a dialog partner (LTERM...,USAGE=D). Via this LTERM partner, a client or a terminal user can connect to the application in order to administer printers and the associated print job queues.

You assign the printers to the respective printer control LTERM via the LTERM partner, i.e. for the LTERM partner you specify the printer control LTERM to which the printers are assigned with LTERM...,CTERM=*printercontrol-ltermname*.

To enable the printer control LTERMs to identify the printers assigned to them, you assign a control identification (CID) to each printer in the PTERM statement. This CID must be unique within the area of a printer control LTERM, because the printer control LTERM addresses the printers using the printer ID. It is particularly important for the printer IDs to be unique in the case of printer groups. Each printer in the pool must be assigned a separate printer ID which does not belong to any other printer in the printer control LTERM.

To restrict access to the printer control LTERM to a particular group of users, you can assign a lock code to the printer control LTERM just like any other LTERM partner.

An acknowledgment procedure is used for the printers assigned to a printer control LTERM. This procedure can be switched on and off as required for each individual printer. All printers assigned to a printer control always run in automatic mode with their first application start after a regeneration.

For further information on printer administration, see the openUTM manual “Administering Applications”.

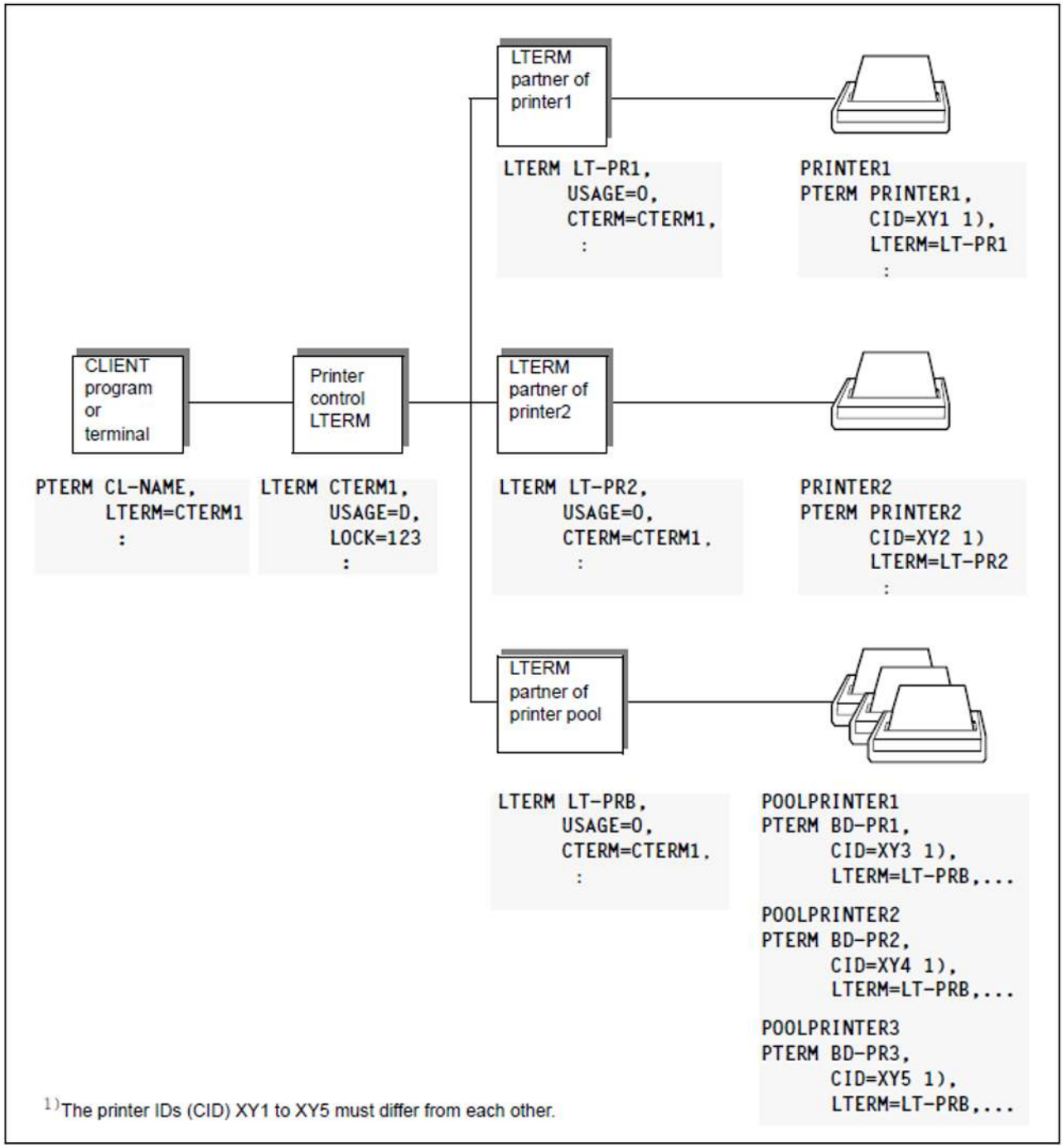


Figure 13: Configuring a printer control LTERM and the associated printers

4.3 Generating service-controlled queues

openUTM offers service-controlled queues, that is, message queues the processing of which is controlled by the program units of the application. A program unit of a dialog or asynchronous service must read the message of a service-controlled queue itself using the KDCS call DGET. A service may also be designed to wait for the arrival of a message.

Since the messages are saved in the page pool, you must ensure that the page pool is configured at a sufficient size.

openUTM provides three different service-controlled queue types:

- USER queues (user-specific)
- TAC queues (defined using TAC statements)
- temporary queues (created using QCRE calls and deleted using QREL calls)

A general introduction to service-controlled queues and their application scenarios can be found in the openUTM manual “Concepts und Functions”.

The implementation of the application scenarios is described in the openUTM manual „Programming Applications with KDCS”. This also contains information about processing service-controlled queues (reading, writing and deleting).

4.3.1 USER queues

Each user of a UTM application is provided with a permanent message queue which is addressed using the user ID.

For USER queues it is possible to generate the data access control which prevents reading or writing by using Q-READ-ACL or Q-WRITE-ACL (USER statement).

USER statement in section "USER - define a user ID"
The following operands are available for USER queues:

- QLEV=

  QLEV can be used to prevent the page pool becoming overloaded with messages for this USER.
  QLEV specifies the maximum number of asynchronous messages that may be buffered in the USER queue (default setting: 32767, i.e. no limit). If the specified value is exceeded, the subsequent behavior is determined by the value in the QMODE parameter.

- QMODE=

  Determines the behavior of openUTM in the event that the USER queue has already exceeded the maximum number of permitted messages that may be buffered and has thus reached the Queue level (operand QLEV=). If the value STD is set all new DPUT calls are rejected, if WRAP-AROUND is set the oldest message is overwritten by the new message.

- Q-READ-ACL=

  Specifies the read and delete authorizations in the USER queue for external users. If you do not specify Q-READ-ACL= all users have read and delete authorization in the queue.

- Q-WRITE-ACL=

  Specifies the write authorization in the USER queue for external users.
  If you do not specify Q-WRITE-ACL= all users have write authorization in the queue.

4.3.2 TAC queues

By generating transaction codes with TYPE=Q (queue) you define permanent Message Queues with fixed names.

The TAC queue with the fixed name KDCDLETQ is called the dead letter queue. openUTM uses this TAC queue to save queued messages sent to transaction codes, TAC queues, LPAP or OSI-LPAPM partners that can not be processed or delivered (see "TAC - define the properties of transaction codes and TAC queues").

TAC queues may be locked for reading or writing (see "Access list concept").

TAC statement in section "TAC - define the properties of transaction codes and TAC queues"
The following operands are important for the generation of a queue defined using TAC statements:

- *tacname*

  Name of the TAC.

- TYPE=Q

  TYPE=Q must be specified for TAC queues. A Message Queue is generated. It is possible to use an FPUT or DPUT call to write a message to a queue of this nature, or to use a DGET call to read a message from the queue.

- ADMIN=

  Specifies whether access to this queue requires administration authorization.

- DEAD-LETTER-Q=

  Specifies whether queued messages in this message queue are to be saved in the dead letter queue if not processed correctly when the maximum number of attempts to redeliver the message (MAX statement, REDELIVERY parameter) has been reached.

- QLEV=

  QLEV can be used to ensure that the page pool is not overloaded by jobs for this TAC queue.
  QLEV specifies the maximum number of asynchronous messages that may be in the Message Queue of this transaction code.

- QMODE=

  Determines the behavior of openUTM in the event that the maximum permitted number of messages is already saved in a queue and thus the queue level is reached.

- Q-READ-ACL=

  The key set defines the authorizations that permit reading or deletion of messages in this queue.

- Q-WRITE-ACL=

  The key set defines the authorizations that permit writing messages to this queue.

- STATUS=

  Specifies whether the message queue is locked or released when the application is started.

4.3.3 Temporary queues

Temporary queues are created and deleted dynamically by program calls. The name of the queue may be determined by the user or assigned by openUTM.

The maximum number of temporary queues is defined within the QUEUE statement.

QUEUE statement in section "QUEUE - reserve table entries for temporary messages queues"
The following operands are used to define temporary queues:

- NUMBER=

  Specifies the maximum number of temporary queues that may exist at any one time during an application run (1 <= NUMBER <= 500 000).

- QLEV=

  QLEV can be used to prevent the page pool becoming overloaded with messages for this temporary queue. QLEV specifies the default value for the maximum number of messages that may exist in a temporary messages queue at any one time (default value: 32767, in other words an unlimited queue length).

- QMODE=

  Determines the behavior of openUTM in the event that the maximum permitted number of messages is already saved in a temporary queue and thus the queue level is reached.

4.3.4 Specifying the maximum waiting time for reading from service-controlled queues

At generation it is possible to set the maximum length of time that a service is permitted to wait for a message for a queue. This maximum wait time may be defined separately for the dialog and asynchronous services (MAX statement, QTIME operand). This ensures that a terminal user or client does not have to wait several minutes for the system to react to an error in a UTM program unit, or that a resource remains blocked for too long.

MAX statement in section "MAX - define UTM application parameters"
The following operands are used to specify the maximum length of time that a service is permitted to wait for a message in a service-controlled queue:

- QTIME=

  Specification of the maximum waiting time in seconds (default: 32767 seconds). You can define separate maximum values for the waiting time for the dialog or asynchronous services.
  If a greater waiting time is specified in a program unit run than specified in QTIME then openUTM resets the waiting time to the value defined here.

4.3.5 Limiting the maximum number of redeliveries to service-controlled queues

At generation you can define whether a message to a service-controlled queue is to be placed back in the queue if the transaction in which the message was read is rolled back. You can also limit the number of redeliveries at generation (MAX statement, REDELIVERY operand). This prevents, for example, endless loops if a program error occurs.

MAX statement in section "MAX - define UTM application parameters"
The maximum number of redeliveries of messages to service-controlled queues is defined using the following operand.

- REDELIVERY= (...,number2)

  number2 is the maximum number of redeliveries of messages to a service-controlled queue (0 <= *number2* <= 255).

  Values between 0 and 254 indicate the number of redeliveries. The value 255 means that the message can be redelivered any number of times.

  Default 255, i.e. the number of redeliveries is unlimited.

4.4 UTM messages

openUTM generates UTM messages that inform about certain events or request dialog input. The UTM messages are located in a message module which is supplied with openUTM (standard message module).

You can modify the messages of openUTM using the message tools KDCMTXT and KDCMMOD, and create own message modules (user message modules) which are adapted to your own needs.

You can adapt the standard UTM messages by:

- modifying UTM message texts (e.g. translation into other languages)
- deleting or adding UTM message destinations
- adding inserts to the message text or remove inserts from it

You must declare user message modules in the configuration using the MESSAGE statement.

The entire event reporting mechanism and the tools KDCMTXT and KDCMMOD are described in detail in the openUTM manual ”Messages, Debugging and Diagnostics”.

4.4.1 Messages in openUTM on BS2000 systems

The following components of UTM event reporting are included in the delivery package:

- the German standard message module KCSMSGS
- the English standard message module KCSMSGSE
- the message definition file SYSMSH.UTM.070.MSGFILE

The message definition file contains the message texts in German and English and forms the basis for the creation of user message modules.

You must declare user message modules in the configuration using the MESSAGE statement. If you do not issue a MESSAGE statement, the German standard message module KCSMSGS is used to output messages

If you want to use the English standard message module in your application, you must replace the German message module with the English message module in the library (see openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”).

In order to internationalize your application you can create multiple user specific message modules in a variety of languages and include them in the configuration of a UTM application. In this way, UTM messages can be output to a terminal user in a variety of

languages within any given UTM application. The language used to communicate with the user depends on the locale (language identifier lang_id and territorial identifier terr_id) that you assign the user during generation as well as on the availability of the user message module which has been assigned an appropriate locale during generation.

If more than one message module is assigned for a UTM application, then a locale must be assigned to each message module.

MESSAGE statement in section "MESSAGE - define a UTM message module"
User message modules are defined with the following operands:

- MODULE=

  Name of the message module you want to incorporate in the configuration.
- LIB=

  Identifies the object module library from which the message module is loaded dynamically. If a generated message module *modulename* is not contained under the name *lmodname* in the library *omlname* when linking the application, the linkage editor reports that the module is missing. The message module can be loaded dynamically.
- LOCALE=

  Defines the language environment (locale) of the message module if language specific message modules have been created for specific message output. These national language message modules are used for users and LTERM partners whose language and territorial identifiers match the locale defined here. For further information, see "Internationalizing the application – XHCS support(BS2000 systems)".

USER statement in section "USER - define a user ID" and
LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
With the following operand you specify the message module (and the language) which is used to output messages to the user/client:

- LOCALE=

  Language environment (locale) of the user/client.

Application message module and user message modules

If multiple message modules are used for an application, then a distinction is drawn between application and user message modules. The application message module is the message module in whose MESSAGE statement the locale specifications correspond to those in the MAX statement. The application message module has a special significance within the application. The message destination specifications entered for the application message module determine the destination for message output. The message destination specifications in the other message modules have no significance. The application message module is used to output messages to the message destinations SYSLST, SYSOUT and CONSOLE.

Messages to the destinations STATION, SYSLINE and PARTNER employ the message module whose *lang_id* and *terr_id* specifications (of the Locale) correspond to those of the user or LTERM partner for which the message is output. Here, the user specification takes priority over the LTERM partner specification, i.e. if a user is signed on when the message is output, openUTM uses the message module which corresponds to this user.

If a locale (*lang_id*, *terr_id*) for which there is no message module in the application has been generated for a user or LTERM partner, then the user or LTERM partner is assigned a message module which corresponds with the *lang_id* and for which no *terr_id* has been generated. If no such message module is present, the application message module is used to output messages to this user or LTERM partner.

MAX statement in section "MAX - define UTM application parameters"
With the following operand you specify the message module which is used as application message module:

- LOCALE=

  The locale of the message module that is to be used as the application message module. A message module with this locale must be generated with a MESSAGE statement.

4.4.2 Messages in openUTM on Unix, Linux and Windows systems

The following components of UTM event reporting are included in the delivery package:

- Standard message module of openUTM

  The standard message module contains the message texts in English and standard settings for the message destinations (e.g. terminals, SYSLOG file).
  openUTM only generates the messages from the standard message module if no NLS message catalogs and no user message modules exist for a language.

  > **CAUTION!**
  > The standard message module must be linked in **each** UTM application program.

  Unix and Linux systems

  The standard message modules `kcsmsgs.o` (K and P messages) and `kcxmsgs.o` (U messages) are supplied with openUTM on Unix and Linux systems:

  - `kcsmsgs.o` is contained in the library `libwork` under the path *utmpath* `/sys`.
  - `kcxmsgs.o` is reloaded from the library `libxmsgs` under *utmpath* `/sys` by default.

  The expression “standard message module“ is used for both modules together.

  Windows systems

  The standard message modules `kcsmsgs.obj` and `kcxmsgs.obj` (U messages) are supplied with openUTM on Windows systems:

  - `kcsmsgs.obj` is contained in the library `libwork.dll` under the path *utmpath* `/ex`.
  - `kcxmsgs.obj` is reloaded from the library `libxmsgs.dll` under *utmpath* `/ex` by default.

  The expression “standard message module“ is used for both modules together.

- Message definition file *msgdescription* (in the *utmpath*)

  It contains the standard message texts in German and English, as well as the framework definitions for the UTM messages (structures of messages).

- NLS standard message catalogs (Unix and Linux systems) / message DLLs (Windows systems)

  NLS standard message catalogs are supplied with openUTM in German and in English.

  On Windows systems the message catalogs are implemented as message DLLs.

  The message catalogs only contain the message texts. When structuring the messages from an NLS catalog, openUTM uses the structure information and message destinations of the default message module, or if available, the user message module.

  Unix and Linux systems

  On Unix and Linux systems, the NLS standard message catalogs are stored in the directories *utmpath* `/nls/msg/` *xxx*. In this case, *xxx* is the language ID for the corresponding language.

  On Unix and Linux systems you can modify existing NLS message catalogs and create your own NLS message catalogs for other languages.

  You can set the language to be used for the messages to your preferred language in the LANG shell variable.

  Windows systems

  On Windows systems, the message DLLs are stored in the directories *utmpath* `\nls\msg\` *xxx*.

  On Windows systems you can change the existing message DLLs and create your own message DLLs for other languages.

  You can set the language to be used for the messages to your preferred language in the LANG shell variable.

In the simplest case, you operate your application with the standard UTM messages, i.e. you do not modify the UTM messages nor the UTM message destinations. In this case, no additional specifications are required in the KDCDEF generation. You must merely link the standard message module *utm-directory/sys/kcsmsgs.o* to the application program.

If you use an own message module, you have to define it via the KDCDEF statement MESSAGE when generating an application. This message module is then created using a C source file created by KDCMMOD.

MESSAGE statement in section "MESSAGE - define a UTM message module"
Use the following operand to define the message module when generating the application:

- MODULE=

  Name of the module that is to be created using the tool KDCMMOD.

To modify the messages, use the message tools KDCMTXT and KDCMMOD (see openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”).

4.4.3 User-specific message destinations

In addition to the message destinations, CONSOLE, SYSOUT etc., there are also four so called user-specific message destinations. The user can define up to four message destinations of their own. These message destinations are named using USER-DEST-*number* and may be user queues, TAC queues, asynchronous TACs or LTERM partners.

> This makes it, among other things, possible to display the K and P messages of your application to the administrator at the WinAdmin or WebAdmin administration workstation (see also in the online help for WinAdmin/WebAdmin, keyword „message collector“).
> Messages indicating warning level violations cannot however always be delivered to their user-specific message destination.

The new KDCDEF statement, MSG-DEST, is used to agree the user-specific message destinations.

MSG-DEST statement in section "MSG-DEST - define user-specific messages destinations"
Using the following operands you can agree a maximum of four user-specific message destinations:

- *msg-destination*

  Refers to the message destination with the specification USER-DEST-*number (number*=1..4*)*. Message destinations must be assigned to the messages using the KDCMMOD.
- NAME=

  Specifies the name of a user or TAC queue or an asynchronous TAC or LTERM partner to which the messages are to be sent (this name must be defined using a TAC, USER or LTERM statement).
- DEST-TYPE=

  Type of message destination (USER queue, TAC or LTERM).

  If you want to forward K and P messages from your application to the WinAdmin or WebAdmin administration workstation, you must specify a TAC queue or a USER queue here.
- MSG-FORMAT=

  Specifies the format of the messages that are to be sent. Only the inserts of a non printable format (FILE; default) or the inserts and messages texts of a printable format (PRINT) are transferred.

Assigning the message destination USER-DEST-*number*

Messages that openUTM is to output to a message destination, USER-DEST-*number*, must also be assigned to this message destination using the utility KDCMMOD (MODMSG statement).

4.5 Message distribution and multiplexing with OMNIS (BS2000 systems)

The services of the BS2000 software product OMNIS can be used for UTM applications on BS2000 systems. OMNIS is a Session Manager that enables a terminal user to call the services of various UTM applications directly, even if the UTM applications are distributed in the network. In this case, the terminal user need not know the processor nor the UTM application in which the service is located. OMNIS automatically establishes a connection to the “correct” UTM application and controls the assignment of messages (message distribution).

When implementing OMNIS, you can also use the multiplex function provided by openUTM on BS2000 systems: a large number of terminals can be connected to a UTM application via a small number of transport connections.

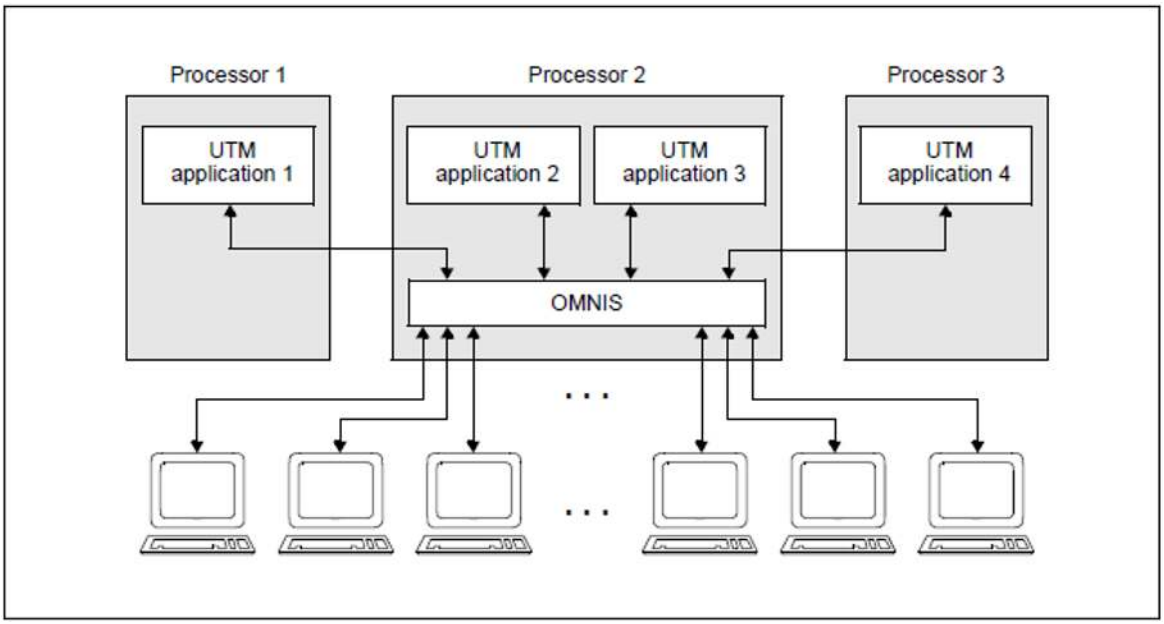


Figure 14: Message distribution and multiplexing with OMNIS

See also the manuals “OMNIS/OMNIS-MENU Functions and Commands” and “OMNIS/OMNIS-MENU Administration and Programming”.

4.5.1 Multiplex connections

In normal dialog mode, one transport connection exists between a terminal and a UTM application on the processor. In order for a user to be able to call the services of an application, the user must open a session with the application, i.e. a communication

relationship between two addressable units in the network. A session setup generally means that the user must provide identification to the application. This can also occur implicitly.

OMNIS now offers you the option of connecting simultaneously to several UTM applications, even on different processors. However, you are only actually connected to one communication partner (namely OMNIS). The Session Manager now transmits the input messages (user jobs) to the applications with which you are connected.

Transport connections and sessions exist on both links of the communication relationship, i.e. the link from user -> Session Manager and from Session Manager -> application. This is illustrated in the diagram below:

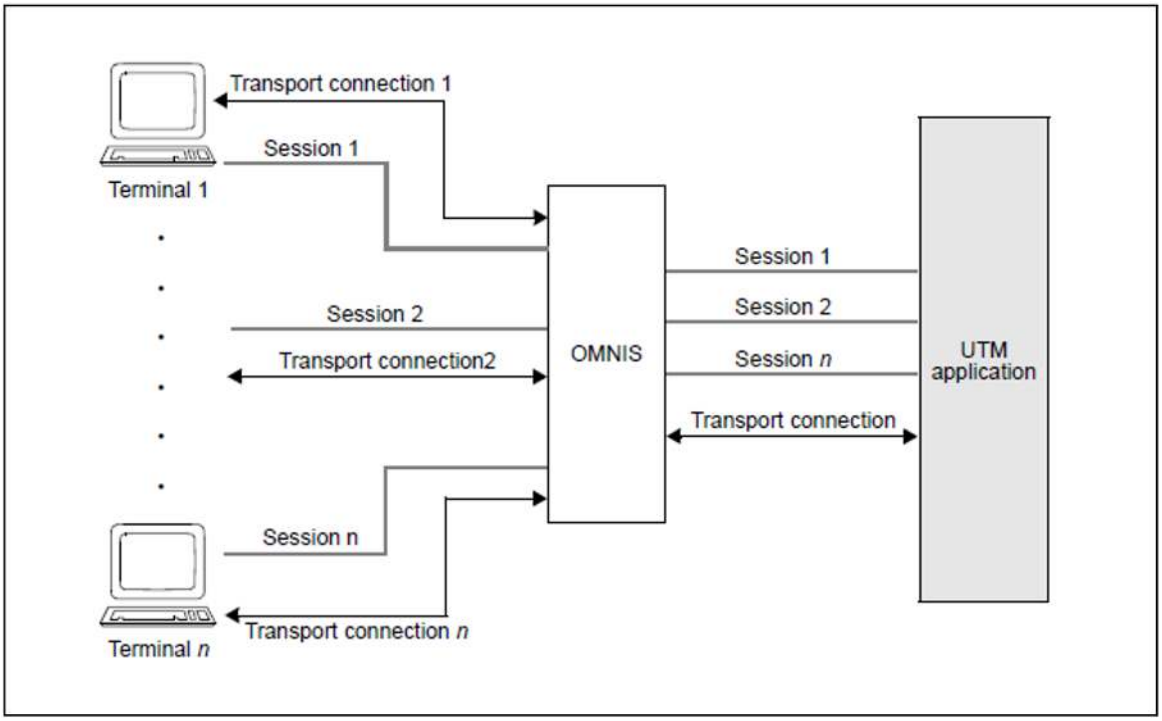


Figure 15: Transport connections and sessions when multiplexing

A **transport connection** is a connection between two programs or between one program and a terminal, via which messages can be exchanged. A transport connection has a defined beginning (connection setup) and a defined end (connection shutdown) and is known to the transport system.

A **session** is one of several completely different data streams, which is maintained via a transport connection. A session has a defined start (session setup) and a defined end (session shutdown) and is known to the transport system. In the special case of OMNIS and openUTM, a session is understood to be a communication relationship between a UTM application and an OMNIS terminal, which begins with the logical opening of the session and ends when the session is closed.

A one-to-one assignment between transport connection and session exists on the link from terminal -> Session Manager.

This one-to-one assignment is cancelled on the link from Session Manager -> application and several sessions can be assigned to a transport connection. In this way, a number of terminals can “multiplex”, i.e. connect to an application via a transport connection. In extreme cases, all sessions between the Session Manager and the application can be processed via a single transport connection.

4.5.1.1 Defining multiplex connections

Each multiplex connection must be described with a MUX statement. It is not possible to enter multiplex connections dynamically.

When multiplexing, the communication between the Session Manager and the application takes place using the PUTMMUX protocol. The task of this protocol is to enable several sessions to be processed via one transport connection and to provide the Session Manager with status information on the UTM application on BS2000 system.

A PUTMMUX connection can exist between a UTM application on BS2000 system and OMNIS as the Session Manager. PUTMMUX connections, also called “multiplex connections”, are defined by the MUX statement when generating the UTM application.

MUX statement in section "MUX - define a multiplex connection (BS2000 systems)"
The most important properties for multiplex connections are defined with the following operands:

- *name*

  Name of the multiplex connection.
- BCAMAPPL=

  Local application name of the UTM application, used by the Session Manager to establish the connection to the UTM application.
- CONNECT=

  Establishment of a transport connection to the Session Manager when the application starts.
- MAXSES=

  Maximum number of simultaneously active sessions between the Session Manager and the UTM application.

When establishing a multiplex connection, openUTM and OMNIS negotiate which MUX protocol versions are supported by both sides of the connection. If there are no MUX protocol versions supported by both partners, the multiplex connection is not established (UTM messages K140 and K141).

The following restrictions apply with the current definition of the protocol:

- connections between two UTM applications are not supported
- printers are not supported
- only the Session Manager can open a session to a UTM application.

The add-on product OMNIS-MENU is available if you are using OMNIS in menu-driven mode. OMNIS-MENU enables you to communicate with various UTM applications via a user-friendly, menu-driven interface. For further details, see the manuals “OMNIS/OMNIS-MENU Functions and Commands” and “OMNIS/OMNIS-MENU Administration and Programming”.

4.5.1.2 Confirming the connection shutdown by the partner

If a user is connected to a UTM application via a multiplex connection, each of the two partners – the UTM application or the user – can request the closedown of this session. As a result of this request, the session switches to the state ”DISCONNECT PENDING”. The session is not yet released. The session is not definitively closed until the partner on the other side confirms the session closedown.

For a specific length of time (approx. 10 minutes) after the request for session closedown has been issued, the session can be released by the closedown confirmation of the partner.Only after this time span has expired can the administrator of the UTM application also release the session (administration command KDCPTERM).

From the output of the administration commands KDCINF PTERM and KDCPTERM, the administrator of the UTM application can determine whether the session is in the state ”DISCONNECT PENDING”. See also the openUTM manual “Administering Applications”.

4.5.2 Statistics on multiplex connections

The administrator of the UTM application can use the command

```
KDCINF MUX,OPTION=MONITORING
```

to instruct openUTM to output statistics on multiplex connections. See also the openUTM manual “Administering Applications”. The UTM administrator receives information on:

- The utilization level of the multiplex connection.

  Information is supplied on the number of input and output messages exchanged via multiplex connections since the start of the application.

- BCAM bottlenecks.

  openUTM supplies information on the number of application messages that could not be accepted by BCAM since the application start due to BCAM bottlenecks, and hence the number of messages openUTM must request be sent again.

4.5.3 Combination of multiplex connections and direct connections

If you are connecting terminals to your UTM application via direct connections as well as via multiplex connections of the Session Manager, the messages are distributed as follows:

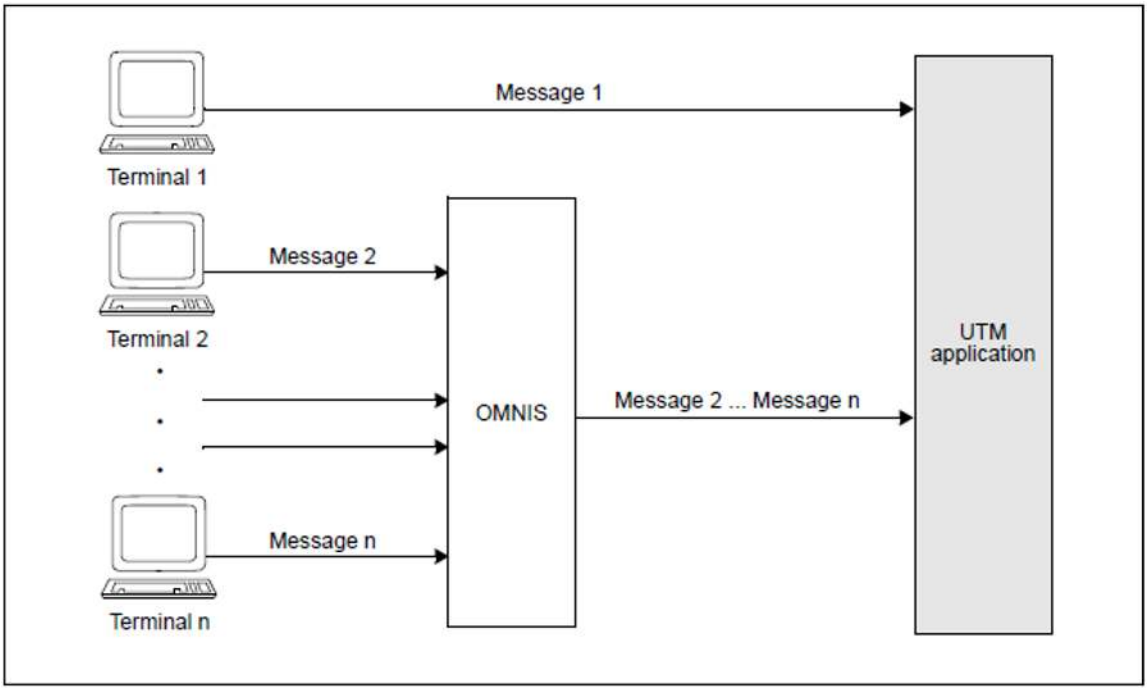


Figure 16: Combination of multiplex and direct connections

This means that messages via direct connections can overtake messages via multiplex connections. In particular load situations, this leads to shorter response times on the direct connection if a data jam occurs on the multiplex connections. There can be several reasons for this:

- The volume of messages from the terminals is so high that the multiplex connections are overloaded.
- All UTM processes are occupied with jobs and therefore cannot retrieve all incoming messages immediately.

There are two ways in which the UTM administrator can avoid the probability of a data jam:

- Increase the number of multiplex connections and distribute the volume of messages evenly over these lines.
- Increase the current number of UTM processes.

To guarantee the administrator the fastest possible access to the UTM application at all times, the administrator’s terminal should be connected to the application via a direct connection.

4.6 Generating load modules, common memory pools and shared code (BS2000 systems)

This section describes how to generate program units, areas and load modules.

In the openUTM manual “Using UTM Applications on BS2000 Systems” you will find more information and recommendations

- on structuring an application program
- on providing shared code in the system memory or in common memory pools
- on the sequence in which modules are loaded and how the external references are resolved
- on program exchange during live operation

4.6.1 Generating load modules

It is only necessary to statically link part of the application to the application program (start LLM, see the openUTM manual “Using UTM Applications on BS2000 Systems”). The other parts of the application program must then be available in the form of dynamically loadable load modules.

As early as the KDCDEF generation you must specify at what point in time you want to load the application parts that are not to statically linked, and to which part of the memory they are to be loaded. You also specify which program units are to be exchangeable during live operation.

The individual load modules of the application must be generated with LOAD-MODULE statements for BLS implementation. You also specify when the module is to be loaded and to where. The sequence with which you generate the load modules determines the sequence in which the load modules are loaded (see LOAD-MODULE statement in section "LOAD-MODULE - define a load module (BLS, BS2000 systems)" and in the openUTM manual “Using UTM Applications on BS2000 Systems”, loading modules).

The assignment of objects (program units and shareable data areas) to load modules is likewise defined in the generation. In the PROGRAM and AREA statements in which program units or shareable data areas are generated, the name you assigned to the associated load module in the LOAD-MODULE statement must be specified in the LOAD-MODULE operand.

> **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 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 generation. Otherwise, openUTM cannot guarantee that a required program will be loaded in the working memory with a particular load module.

The load modules are described at generation in the following manner:

LOAD-MODULE statement in section "LOAD-MODULE - define a load module (BLS, BS2000 systems)"

The properties for load modules are defined with the following operands:

- *lmodname*

  Name of the load module. This name is used to assign objects to load modules during generation (program units, areas).

  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.

- LOAD-MODE=

  Specifies when a load module is to be loaded, and specifies the memory area to which it is to be loaded. The load modules can be loaded in the standard context to the local task memory, to a common memory pool or to the system memory.

  The parts of the application program can be:

  - Linked statically to the application program (LOAD-MODE=STATIC)

    The part of the application program that is loaded to the standard context of the application using the command `START-EXECUTABLE-PROGRAM` or `LOAD-EXECUTABLE-PROGRAM .`

  - Dynamically loaded to the standard context of the local task memory when the application is started (LOAD-MODE=STARTUP).

    These should be program units that are continuously required by the UTM application, or which contain external references to shareable parts of the application.

  - Loaded to the standard context of the local task memory at the first call (LOAD-MODE=ONCALL)

    These should be program units that are not continuously required by the application.

  - Loaded to a common memory pool (LOAD-MODE=(POOL,*poolname,*...))

    The common memory pool must be generated with a MPOOL statement (see "Shared code in common memory pools").
    The program units that should be loaded to the common memory pool are those that are required by all processes of a UTM application, and which are shareable, for example, the shareable parts of your program unit or also formats or data areas.
    If an LLM contains public and private slices, the public slice is loaded in a common memory pool and the private slice is loaded in the standard context in the local task memory. You can specify whether the non-shareable part is to be loaded when the application is started (LOAD-MODE=(POOL, pool name, STARTUP)) or only then when that program unit is called (LOAD-MODE=(POOL, pool name, ONCALL)).
    For more information about the generation of shared code see also "Generating shared code and common memory pools".

  - Loaded to the system memory as a non-privileged subsystem.

    These application parts must be loaded to the system memory by the BS2000 system administrator before the application is started.

    The private slice of a shareable part contained in nonprivileged subsystems can be linked to the static part of an application program, either when the application is started or the first time it is called.
    How to generate the non-shareable parts is described in the section "Shared code in system memory".

- LIB=

  Specifies the library from which the load module is to be loaded

  You can specify object module libraries (OML) or program libraries (PL) which contain type R or L elements.

- VERSION=

  Specifies 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 specification of the version to define which version of an element is to be loaded.

- ALTERNATE-LIBRARIES=

  Specifies 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.

  openUTM suppresses the BLS autolink function when loading dynamically and when exchanging programs, if you specify ALTERNATE-LIBRARIES=NO. The load module then must only have open external references to program components that already exist in the working memory when this module is loaded. For load modules that are generated using POOL or STARTUP, the sequence of the LOAD-MODULE statements at generation is critical for the resolving of open external references at loading. The sequence with which you generate the load modules determines the sequence in which the load modules are loaded.

  ALTERNATE-LIBRARIES=YES ensures that runtime system modules that are also required are dynamically linked when an exchange is made. The autolink function may 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.

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.

The assignment of objects (PROGRAM, AREA statement) to load modules (LOAD-MODULE statement) is also defined in the generation.

AREA statement in section "AREA - define additional data areas"
PROGRAM statement in section "PROGRAM - define a program unit"

The assignment to load modules is defined the following operand:

- LOAD-MODULE=

  Name of the load module (*lmodname* in the LOAD-MODULE statement), to which the program is linked.

  Program units, modules, and data areas must be linked statically to the application program if the load module to which they are assigned was generated with LOAD-MODE=STATIC or if they are not assigned to any load module.

  The administration modules (e.g the KDCADM administration program) 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 (LOAD-MODE=STARTUP). The same applies to the START, SHUT, INPUT and FORMAT event exits and the BADTAC, MSGTAC and SIGNON event services.

If specifications for objects in the statements AREA, LOAD-MODULE, MPOOL, PROGRAM and TAC are modified in the generation, only one new KDCFILE need be created. The next application start must then be based on the new KDCFILE.

4.6.2 Generating shared code and common memory pools

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.

> ¡ If parts of a program unit are to be shareable, this must be taken into account in the programming. For further information, see the openUTM manual „Programming Applications with KDCS” or the appropriate language supplement.

4.6.2.1 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.

The shareable modules must be loaded in the memory by the administrator before the application is started. They can be exchanged while the application is running.

Non-shareable parts of the program units must be created as follows:

- The entry point of the program unit (it is in the non-shareable part or in the private slice) must be described in a PROGRAM statement and assigned to a load module there using the LOAD-MODULE operand in the PROGRAM statement.
- The load module must be generated with a LOAD-MODULE statement with LOAD-MODE={STARTUP | ONCALL}. The load module or its private slice is loaded dynamically into the local task memory (class 6 memory) at the start of the application program. The links in the shared code are established dynamically using the external references to the shareable modules.

The load modules (OM format) containing the shareable modules of the program unit and the load modules containing the non-shareable program components must not occur together in a program library.

Example

```
PROGRAM NONSHARE,LOAD-MODULE=NAME1,COMP=ILCS
LOAD-MODULE NAME1,LIB=UTM.PLIB,LOAD-MODE=STARTUP,VERSION=001
```

NONSHARE is located in the non-shareable part (for LLMs in the private slice) of the program unit.

4.6.2.2 Shared code in common memory pools

Objects that are not linked statically when linking the application program can be loaded into a common memory pool. In a common memory pool you can dynamically load several load modules.

A common memory pool must be generated with the KDCDEF statement MPOOL.

MPOOL statement in section "MPOOL - define a common memory pool (BS2000 systems)"
The most important properties for common memory pools are defined with the following operands:

- *poolname*

  Name of the common memory pool. This name is used at generation to assign to a pool those load modules whose Public Slice is to be loaded to the pool (see LOADE-MODULE statement).

- SCOPE=

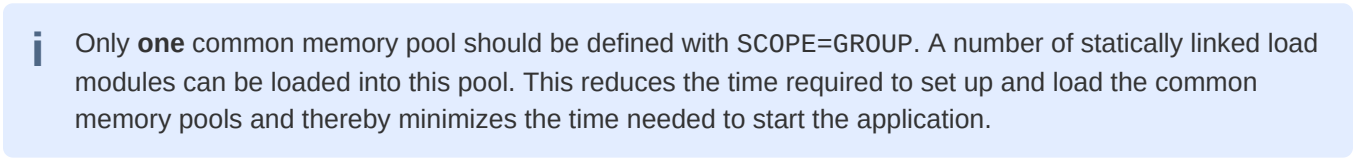
  Specifies the scope of the pool (local application with SCOPE=GROUP or global application with SCOPE=GLOBAL).
  For each BS2000 user ID, BLS supports a maximum of eight common memory pools with SCOPE=GROUP and eight common memory pools with SCOPE=GLOBAL.

- PAGE=

  Hexadecimal address in the form X'xxxxxxxx'.
  If global common memory pools with the same contents/names are used in several UTM applications, the parameter PAGE=X'xxxxxxxx' must be specified with the same address in all applications. The address entered using PAGE= is to be selected in such a way that the address area reserved is available in all these applications.

- SIZE=

  Specifies the size of the common memory pool.
  The size is specified in units of 64 KB. With 24-bit addressing, the size of a common memory pool is always a multiple of 64 KB. With 31-bit addressing, the size of the common memory pool is calculated by *n* * 1MB >= SIZE * 64 KB (where *n* is selected as a minimum).



Only **one** common memory pool should be defined with 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.

Generating shareable objects that are to be loaded in a common memory pool

The following section describes how you generate shareable objects that are to be loaded in a common memory pool if you are working with BLS.

- 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. 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*, and the private slice is loaded dynamically either when the application starts (STARTUP) or when the program is called (ONCALL). Additional PROGRAM statements are required for the programs of these LLMs that are called by openUTM.

  If a compiler created two separate object modules for the shareable and non-shareable part, then should link these modules beforehand to an LLM with slices using the linker. You can then generate this LLM as described above.

  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 which 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 linker can also be linked together beforehand with other modules in one LLM with slices. This LLM can be generated in the following manner:

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

Example

The example assumes that the COBOL85 compiler was used for compiling and that the compiler has saved the objects in an LLM.

The shareable modules of the COBOL program units PU1 and PU2, and the data module DATAMOD are to be loaded in the local application pool LCPOOL. LCPOOL is to be loaded at address X'020000', occupy 128 KB, and 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 statically linked to the LLM-LCPOOL LLM before the application is started, i.e. you must specify the option BY-ATTRIBUTES(PUBLIC=YES) in the BINDER statement START-LLM-CREATION, whereby the LLM is divided into a public slice and a private slice. The LLM created in this way must be made available in the library *libname*.

4.7 Code conversion

During communication between the UTM application and a client or a partner application on a different platform, it is possible that the two communication partners will be using different codes, as Unix, Linux and Windows systems use ASCII-compatible codes and BS2000 systems use an EBDDIC code. To simplify communication between the partners, you can initiate automatic code conversion for clients and partner applications via generation:

- BS2000 systems: TS applications of type SOCKET and HTTP clients
- Unix, Linux and Windows systems:
  - OpenCPIC clients and TS applications of type SOCKET and APPLI.
  - Server-server communication with LU6.1 and OSI TP partners

You must remember that only printable messages may be exchanged, as binary data may become errored if converted.

You can use up to four different code conversion in a UTM application. openUTM provides code conversion tables for this purpose.

The code conversion is controlled using the operand MAP of the PTERM, TPOOL, OSI-CON and SESCHA statements, and - for HTTP clients - by use of the statements CHAR-SET and HTTP-DESCRIPTOR.

```
PTERM/TPOOL ... MAP= USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4
OSI-CON ... MAP = USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4
SESCHA ... MAP = USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4
CHAR-SET   SYS1 | SYS2 | SYS3 | SYS4, ....
HTTP-DESCRIPTOR ..., CONVERT-TEXT = *YES
```

Code conversion tables

openUTM converts the data using the code conversion tables provided. These tables convert the data as follows:

BS2000, Unix- and Linux systems:

- SYS1/SYS/SYSTEM: ISO8859-i <-> EBCDIC.DF.04.i (EDF04i)
- SYS2: ISO8859-1 <-> EBCDIC.DF.04.DRV (EDF04DRV)
- SYS3: ISO646-IRV <-> EBCDIC.03.DF.03.IRV (EDF03IRV))
- SYS4: ISO646-IRV <-> EBCDIC.03.DF.03.DRV (EDF03DRV).

Windows systems:

- SYS1/SYS/SYSTEM: Windows-1252 <-> EBCDIC.DF.04.F (EDF04F)
- SYS2: Windows-1252 <-> EBCDIC.DF.04.DRV (EDF04DRV)
- SYS3: ISO646-IRV <-> EBCDIC.03.DF.03.IRV (EDF03IRV))
- SYS4: ISO646-IRV <-> EBCDIC.03.DF.03.DRV (EDF03DRV).

The first code conversion is referred to below as the standard code conversion.

The first and second code conversion are conversions between two 8-bit codes. The third and fourth code conversion are conversions between two 7-bit codes.

The code conversion tables provided can be modified or replaced with custom tables. In this guide, conversion between an EBCDIC code and an ASCII code is always assumed, although theoretically you can also convert between any EBCDIC codes.

The conversion tables are located:

- In the assembler source KDCEA on BS2000 systems,
- In the C source kcsaeea.c on Unix, Linux and Windows systems.

The sources contain eight conversion tables for the four code conversions.

Additional information about code conversion can be found in the openUTM manual „Programming Applications with KDCS”. There you also find hints how to operate modified code tables or own code tables.

4.8 Job control - priorities and process limitations

openUTM provides two methods with which you can control the distribution of released UTM processes amongst the jobs ready for processing. This means that you can affect the order in which openUTM starts the processing of jobs on transaction codes.

By using one of the methods for job control, you can:

- give important jobs higher processing priority
- prevent many jobs of the same type from running at the same time, thereby causing the processing of other jobs to be delayed
- prevent the blocking of job processing due to long-running jobs. Long-running jobs are services whose processing takes an extremely long time, e.g. because their program units are searching through data or they contain program waits (blocking calls such as PGWT).
- in UTM cluster applications, prevent too many tasks from simultaneously accessing memory areas that are available globally in the cluster.

With both methods you must assign TAC classes to the transaction codes that are subject to a specific job control. You can then **alternatively** select one of the two methods for job control between TAC classes:

- Priority control
  The distribution the processes amongst the TAC classes is controlled by priorities. These priorities are used by openUTM to determine when the outstanding jobs are to be processed. You turn priority control on with the KDCDEF statement TAC-PRIORITIES.
- Process limitations
  You limit the number of the processes that are allowed to process jobs of a certain TAC class simultaneously, or you specify how many processes are to remain free for processing jobs of other TAC classes. The process number can be specified individually for every TAC class. The KDCDEF control statement TACCLASS is provided for specifying the number of processes.

You cannot use the two methods together in an application, i.e. you must not use the control statements TAC-PRIORITIES and TACCLASS together in the KDCDEF generation.

Assigning transaction codes to TAC classes

openUTM differentiates between a total of 16 TAC classes. There are 8 classes each available for dialog and asynchronous transaction codes, classes 1 through 8 for dialog transaction codes and classes 9 through 16 for asynchronous transaction codes.

You specify the assignment of the transaction codes to the TAC classes in the KDCDEF generation.

TAC statement in section "TAC - define the properties of transaction codes and TAC queues"
Operand TACCLASS=

openUTM makes the following assignments for transaction codes to which you have not explicitly assigned a TAC class (no entry in TACCLASS=):

- dialog transaction codes are not assigned to a TAC class
- asynchronous transaction codes are assigned to TAC class 16

You should combine the transaction codes of similar types of services into one TAC class. A TAC class then represents a type of job in your application.

Which jobs are subject to job control?

Generally only jobs that have been placed in a job queue by openUTM are subject to job control.

Jobs for asynchronous transaction codes are always placed in a job queue first before openUTM selects them for processing.

Jobs for dialog transaction codes, on the other hand, are only placed in a queue in bottleneck situations, e.g. when the number of the available processes has been exhausted.
If the load on the application is low, then the dialog jobs are processed immediately because they will not block each other significantly and buffering in the queue would appear to slow the system.

For this reason, the methods for job control for asynchronous jobs are always used, while the methods for job control for dialog jobs are only used in bottleneck solutions.

In addition, the following jobs are not subject to job control:

- Jobs for dialog transaction codes that are not assigned a TAC class. These jobs are always started immediately after they have been received from the transport system.
- Jobs for the transaction codes KDCSGNTC, KDCMSGTC and KDCBADTC with which the event services (sign-on service, MSGTAC and BADTACS program) are started.

Distribution of resources amongst dialog, asynchronous and PGWT processing

In an initial stage of job processing you should - regardless of the methods used for job control - specify the maximum number of processes of the application that are allowed to process asynchronous jobs at the same time or to wait in Program Wait at the same time. In this manner you can prevent the dialog operation of your application from slowing down due to the processing of such jobs.

MAX statement in section "MAX - define UTM application parameters"
The process numbers are with the following operands generated:

- ASYNTASKS=(*atask_number*,...)

  With *atask_number* you specify the maximum number of the processes of the application that may simultaneously process jobs for asynchronous TAC classes.
- TASKS-IN-PGWT=

  The maximum number of processes of the UTM application in which program units with blocking calls are allowed to run simultaneously. You must specify TASKS-IN-PGWT > 0 if you want to assign the PGWT=YES property to transaction codes or TAC classes.

The values specified for ASYNTASKS=(*atask_number*,...) and TASKS-IN-PGWT in the MAX statement are maximum values. When starting the application and in application mode, you can lower the number of processes via the administration to adapt to the current situation.

Default setting

If you do not create any TAC classes, i.e. you do not specify the TACCLASS operand in the TAC statement, then openUTM does not perform any special job control.
Program unit runs with blocking calls are not allowed then. Dialog jobs are processed in the order in which they arrive in openUTM.

If you do not issue a TACCLASS or a TAC-PRIORITIES statement in the generation, then openUTM automatically applies the methods used to limit the number of processes. All TAC classes are administrable, i.e. the UTM administrator can specify numbers of processes for the TAC classes.

4.8.1 Job processing via priority control

To activate job control via priorities you must issue the TAC-PRIORITIES statement in the KDCDEF generation. In it you also specify the algorithms with which the individual dialog or asynchronous TAC classes are to be prioritized.

TAC-PRIORITIES statement in section "TAC-PRIORITIES - specify priorities of the TAC classes"
You specify the algorithms for the priority control with the following operands:

- DIAL-PRIO=
  Priority with which the available processes of the application are to be distributed amongst the dialog TAC classes.
- ASYN-PRIO=
  Priority with which processes for the asynchronous TAC classes with ready asynchronous jobs or interrupted asynchronous jobs are to be distributed.

You can select between the **absolute**, a **relative** or the **same** priority for both dialog and asynchronous TAC classes.

The following is always true, regardless of which algorithm you select:

- The TAC class 1 of the dialog TAC classes has a higher or the same priority as TAC class 2, and this has a higher or the same priority as TAC class 3, etc.
- For asynchronous TAC classes, class 9 has a higher or the same priority as TAC class 10, and this has a higher or the same priority as TAC class 11, etc.

If **absolute priority** is selected, then free processes of the application are always assigned the TAC class with the highest priority, meaning 1 (dialog) or 9 (asynchronously) as long as there are jobs waiting for this TAC class. Only after there are no more jobs waiting in the TAC class with the highest priority are waiting jobs of the TAC class with the next lower priority processed. When the load is high, absolute priorities leads to waiting jobs of a TAC class with a lower priority not being processed for a long time. If you want to prevent this, then you should use relative priorities.

If **relative priority** is selected, then jobs from TAC classes with higher priorities are processed more often than jobs from TAC classes with lower priorities, i.e. free processes are more often assigned higher priority TAC classes (e.g. 1) than lower priority TAC classes if there are jobs ready and available for this. If there ar e jobs available for all TAC classes, then class 1 is serviced twice as often as class 2, and class 2 is serviced twice as often as class 3 (and so on). The same is true for asynchronous TAC classes.

If **same priorities** is selected, then the same number of jobs (if there are any) from every TAC class are processed.

Jobs within the TAC classes, however, whose processing leads to program waits (TACs with PGWT=YES) are only processed if the maximum number of processes allowed to process the PGWT jobs has not yet been reached.

Reserving processes for dialog jobs outside of the TAC classes

When using priority control for the TAC classes, you can limit the number of processes that process the jobs of the TAC classes to keep some processes free for administrative tasks or internal UTM jobs.
This limitation is the same, however, for all asynchronous TAC classes and for all dialog TAC classes.

You limit the maximum number of processes for asynchronous TAC classes with
MAX ASYNTASKS=(*atask_number*,...) as described in chapter "Job control - priorities and process limitations".

You limit the number of processes for the dialog TAC classes with the FREE-DIAL-TASKS= operand of the TAC-PRIORITIES statement.

The number of processes specified in FREE-DIAL-TASKS is reserved for the processing of jobs that do not belong to any dialog TAC class. These jobs are asynchronous jobs and dialog jobs that are not assigned a dialog TAC class, and in particular are internal UTM tasks (establishing connections, sending acknowledgments, starting the MSGTAC routine, etc.). One of the internal UTM tasks is to pick up the incoming jobs for the UTM application at the job market and, if necessary, enter these in the job queues of the application. These “reserved processes“ then help to offload the job market. In particular, if many jobs sent to the application come from the network, then this will prevent a backlog in the network that may reach all the way back to the communication partner.

The number of processes you should reserve for this task depends on your application. It is recommended to reserve one or two processes for this task.

You can change the number of free processes via the administration.

See the openUTM manual “Administering Applications”; KDCADMI operation code KC_MODIFY_OBJECT with object type KC_TASKS_PAR

Example

The following maximum number of processes is specified in the KDCDEF generation:

```
MAX TASKS=7,ASYNTASKS=2
TAC-PRIORITIES ...,FREE-DIAL-TASKS=3
```

If the application is then started with six processes (start parameter TASKS=6), then the following process numbers are available:

- Three processes for processing jobs for the dialog TAC classes 1 through 8 (determined by: TASKS – FREE-DIAL-TASKS = 6 – 3 = 3)
- Two (=ASYNTASKS) processes for processing jobs for the asynchronous TAC classes 9 through 16
- One process for internal UTM tasks and dialog jobs for transaction codes that are not assigned any TAC class (determined by: FREE-DIAL-TASKS – ASYNTASKS = 3 – 2 = 1)

For information on the use of TAC priorities in UTM cluster applications, see also the applicable openUTM manual “Using UTM Applications on Unix, Linux and Windows systems”, section "Using global memory areas" in the chapter "UTM cluster applications".

4.8.2 Job processing via process limitation for TAC classes

Job control via process limitation is generated using the TACCLASS statement. Process limitation depends on the TAC classes, i.e. you can issue a separate TACCLASS statement for every TAC class.

TACCLASS statement in section "TACCLASS - define the number of processes for a TAC class"
You can alternatively specify one of the two following operands to set up process limitation:

- TASKS=

  The maximum number of processes that are allowed to process jobs for this TAC class.
- TASKS-FREE=

  The minimum number of processes that are to be kept free for the processing of jobs from other TAC classes or of jobs that are not assigned a TAC class.

In this method the number of the TAC class says nothing about the priority with which its jobs are processed. Only the number of processes that you allow for this TAC class specifies how strongly the processing of the jobs is suppressed as compared to other TAC classes.

This method can then be used sensibly when only a few different types of jobs (and therefore only a few TAC classes) in a application and, for example, when you want to prevent long-running jobs from reserving all the processes of an application and therefore unnecessarily slowing down the processing of other important jobs, e.g. administration jobs.

For information on the use of TAC classes in UTM cluster applications, see also the applicable openUTM manual “Using UTM Applications on Unix, Linux and Windows systems”, section "Using global memory areas" in the chapter "UTM cluster applications".

4.8.3 Comparison of some of the properties of the two methods

You can only use one of the two methods for job control in your UTM application. Which of the two possibilities you should select for your application also depends on the sometimes different properties of the two methods.

Program units with blocking calls

- *Priority control*

  Transaction codes from program units that execute blocking calls may be assigned any TAC class as long as a value > 0 is generated in the TASKS-IN-PGWT operand of the MAX statement. You must specify the operand PGWT=YES in the TAC statement for transaction codes with blocking calls.

  `TAC ..., TACCLASS=`*number*`,PGWT=YES`

  This also allows you to process corresponding jobs with different priorities.

- *Process limitation*

  All transaction codes from program units that execute blocking calls must be assigned the same dialog or asynchronous TAC class. You must generate these dialog or asynchronous TAC class as follows:

  `TACCLASS ...,PGWT=YES`

  The corresponding dialog or asynchronous jobs are thus handled in the same way.

Temporarily stopping the execution of certain asynchronous jobs

Both methods for job control provide a mechanism with which you can temporarily prevent the processing of certain asynchronous jobs. These jobs are then received and accepted by openUTM and written in the message queue of the corresponding transaction code. The processing of these jobs is only initiated after the “processing lock“ is removed by the UTM administration.

To temporarily prevent the execution of jobs, set the status of the transaction code to KEEP. You can do this during live operation via the UTM administration or do this during the generation of the transaction codes by specifying the following:

`TAC ...,STATUS=KEEP`

openUTM processes the buffered jobs first when you set the status of the transaction code to ON.

See the openUTM manual “Administering Applications”; KDCADMI operation code KC_MODIFY_OBJECT with object type KC_TAC or the administration command KDCTAC

When using the *process limitation* method, the execution of jobs can also be prevented for all transaction codes of an asynchronous TAC class. In this case you must set the maximum number of processes that are available for jobs of this TAC class to 0.

`TACCLASS ...,TASKS=0`

openUTM only processes the jobs if you increase the maximum number of processes again.

See the openUTM manual “Administering Applications”; KDCADMI operation code KC_MODIFY_OBJECT with object type KC_TACCLASS or the administration command KDCTCL

You can use both mechanisms, for example, to collect jobs that are to be executed at a later point in time when the load on the application is lower (e.g. at night).

In both cases you should limit the message queue of the transaction code(s) to prevent overloading the page pool with too many buffered jobs. This is done for each TAC by:
`TAC ...,QLEV=`

Change of process when processing jobs

- *Priority control*

  If a service consists of several program units (follow-up TAC after a PEND PA/PR), then a change of process can always occur when processing the service, regardless of whether the current TAC and follow-up TAC belong to the same TAC class or not.

- *Process limitation*

  For job control via process limitation, openUTM guarantees that no change of process will occur after a PEND PA /PR and SP when the service TAC and follow-up TAC are assigned the same TAC class.

  If the current TAC and follow-up TAC belong to different TAC classes, then a change of process can also occur when using this method.

Change of process for asynchronous services

When a change of process occurs, an asynchronous service is inactive at first and does not reserve a UTM process although it remains open.

You can limit the maximum number of simultaneously open asynchronous services. You must specify the following in the MAX statement to do this:

MAX ...,ASYNTASKS=(...,*service_number*).

If *service_number* of open asynchronous services exist, then no new asynchronous job that is ready is started. An interrupted open asynchronous service is selected from the next process that becomes free, and this service is resumed.

4.8.4 Process priorities on BS2000 systems

openUTM uses the methods described above for job control to select a job that is to be restarted or resumed. Jobs that are currently being processed cannot be influenced with these methods.

You can use the scheduling mechanisms of BS2000 systems for prioritizing jobs to influence the priority of the active jobs. The RUNPRIO operand of the TAC statement can be used for this purpose. With RUNPRIO you assign a transaction code a process priority (Run-priority) in the KDCDEF generation. You can influence the speed with which a running job is processed with the process priority. A job for a transaction code with a higher process priority will be given preference when distributing the CPU resources in comparison to other jobs with lower priorities.

If you have generated a process priority for a transaction code, then openUTM sets the BS2000 process priority of the process that is processing a job for this transaction code to the value generated in RUNPRIO.
You can specify a value between 30 (highest priority) and 255 (lowest priority) in RUNPRIO.

TAC statement in section "TAC - define the properties of transaction codes and TAC queues" operand RUNPRIO

4.9 Authorization Concept

When you have services that access security-relevant data, it is sensible to restrict access to a limited number of authorized users. openUTM offers two possible methods of data access control which allow you to set different data access authorizations in a UTM application:

- access list concept (service-oriented)
- lock/key code concept (user-oriented)

Both processes use, for the most part, the same generation interfaces.
The greatest difference lies in the way in which the UTM objects are seen: The access list concept allows you to specify a list of codes for each service. These codes specify which user (types) are permitted to access the data. The lock/key code concept allows you to define an (individual) lock code for each service and then assigns each user the appropriate key codes.

Services whose TACs are not secured by a lock code or access list can be called by all users without restriction.

For detailed information about the access list and the lock/key code concepts see the openUTM manual “Concepts und Functions”.

4.9.1 Lock/key code concept

A lock code is a number which symbolizes a logical lock. The objects that are to be protected - for example, the LTERM partner and the transaction codes assigned to the services - are assigned a lock code (TAC or LTERM statement).
Key codes are defined for user IDs and for LTERM partners (USER or LTERM statement). Only when the key code corresponds to the lock code of a protected object is access to this object permitted.
Since a user ID or LTERM partner usually has access to several services, they must also have several key codes. The individual key codes are thus organized into key sets (KSET statement).

The lock/key code concept has the following significance:

- It is only possible to sign on under a UTM user ID if the specified user ID is assigned a key code which corresponds to the lock code of the LTERM partner via which sign-on is performed.
- A user can only call a service when **both** the key set of the current (UTM) user ID **and** that of the LTERM partner contain a key code that corresponds to the lock code of the transaction code.

KSET statement in section "KSET - define a key set"
You can use the following operands to define a key set.

- *keysetname*

  Name of the key set.
- KEYS=

  When assigning a key set to a user (USER):
  Specification of one or more key codes (numeric) that are assigned to the user.

  When assigning a key set to an LTERM partner (LTERM):
  Specification of one or more key codes (numeric) that are assigned to the LTERM partner.

TAC statement in section "TAC - define the properties of transaction codes and TAC queues"
You can use the following operands to control access to the TAC.

- *tacname*

  Name of the TAC.
- LOCK=

  Specifies the lock code that is assigned as a form of logical combination lock to the TAC of a service.

  A service that is protected by a lock code can only then be started if the key set of the user **and** the key set of the LTERM partner both contain a key code that corresponds to the lock code.

  This operand may not be specified in conjunction with the operand ACCESS-LIST=.

USER statement in section "USER - define a user ID"
You can use the following operands to assign a key set to a user.

- *username*

  UTM user ID.

- KSET=

  Specifies the name of the key set that is assigned to the user ID. The key set must be defined using the KSET statement. A maximum of one key set can be assigned to a user.
  A user is only able to access a service whose first TAC is protected by a lock code if one of the key codes in the key set of the user corresponds to the lock code. Otherwise access to the service is denied.

LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
TPOOL statement in section "TPOOL - define an LTERM pool"
You can use the following operands to assign a key set to an LTERM partner.

- *ltermname*

  Name of the LTERM partner (only for LTERM statement).

- LTERM= , NUMBER=

  Name of the LTERM partner (only for TPOOL statement).

- KSET=

  Specifies the name of the key set that is assigned to the LTERM partner. The key set must be defined using the KSET statement. For the LTERM partners of a UPIC client or a TS application without an explicitly generated connection user ID this key set is also the key set of the connection user ID.

- USER-KSET= (only for TPOOL statement)

  In LTERM pools for TS applications or UPIC clients this specifies the name of the key set that is assigned to the connection user ID. This key set must be defined using the KSET statement. The access authorizations are derived from the intersection of the key sets from KSET= and USER-KSET=.

- LOCK=

  The lock code that is assigned to the LTERM partner as the logical combination lock. Only valid for clients (USAGE=D).

  Only a (UTM) user for whom a key set has been generated with a key code that matches the lock code of the LTERM partner can sign on to the application via an access-controlled LTERM partner.

4.9.2 Access list concept

An access list is a number of access codes (numeric codes) that are assigned to a service. The access codes in the access list defines user access to a service and can be interpreted as the roles of the users within the structure of their organization (for example, general users, heads of department, system administrators).
If you use the administration tool WinAdmin or WebAdmin you can use meaningful names in place of numeric codes.

An access list is defined using the KSET statement and assigned to a service using the TAC statement. The roles for the user (USER) are also defined and assigned as a key set using a KSET statement. In the same way, it is also possible to assign an LTERM partner a certain number of roles.

A user can only access a service (TAC) protected in this way if both the key set of the user and the key set of the LTERM partner via which the user has signed on contains at least one of the roles that are contained in the access list of the service.

The differences between the lock/key code and the access list concepts are described in detail in the security function section of the openUTM manual “Concepts und Functions”.

KSET statement in section "KSET - define a key set"
The following operands can be used to define key sets or access lists.

- *keysetname*

  Name of the key set or access list.

- KEYS=

  When assigning an access list to a service (TAC):
  Specification of one or more roles (as numerical values) that have access to the service protected by *keysetname*.

  When assigning a key set to a user (USER):
  Specification of one or more roles (as numerical values) that are to be assigned to the user.

  When assigning a key set to an LTERM partner (LTERM):
  Specification of one or more roles (as numerical values) that may be performed when signing on via this LTERM partner.

  > When using WinAdmin or WebAdmin you may also assign roles with alphanumeric names.

TAC statement in section "TAC - define the properties of transaction codes and TAC queues"
The following operands are used to control the accesses to the TAC.

- *tacname*

  Name of the TAC.

- ACCESS-LIST=

  Specifies the access list that controls access to this TAC. Only users whose key set contains at least one of the roles contained in this access list and that sign on via a terminal that has also been assigned one of these roles may access this TAC. ACCESS-LIST may not be specified in conjunction with LOCK.

USER statement in section "USER - define a user ID"
The following operands are used to assign a key set to a user.

- *username*

  UTM user ID.

- KSET=

  Specifies the name of the key set that the user ID is assigned to. The key set must be defined using the KSET statement. Each user can be assigned a maximum of one key set.
  If a user wishes to access a service that is protected with an access list then at least one of the roles of the user must be contained in the access list. Otherwise access to the service will be denied.

LTERM statement in section "LTERM - define an LTERM partner for a client or printer"
TPOOL statement in section "TPOOL - define an LTERM pool"
The following operands are used to assign a key set to an LTERM partner.

- *ltermname*

  Name of the LTERM partner (only for LTERM statement).

- LTERM= , NUMBER=

  Name of the LTERM partners (only for TPOOL statement).

- KSET=

  Specifies the name of the key set assigned to the LTERM partner. For the LTERM partner of a UPIC client or a TS application without explicitly generated connection user ID this key set is the same as the key set of the connection user ID. The key set must be defined using the KSET statement. Each LTERM partner may be assigned a maximum of one key set.

- USER-KSET= (only for TPOOL statement)

  In LTERM pools, specifies the name of the key set for TS applications or UPIC clients that is assigned to the connection user ID. The key set must be defined using the KSET statement. The access authorizations are derived from the intersection of the key sets of KSET= and USER-KSET=.

  > Access to the LTERM partner may not be protected using access lists. When using access lists to provide data access control to services, you should not use access protection on the LTERM partner, or in other words the parameter LOCK of the LTERM and TPOOL statements may not be specified.

Data access control for service-controlled queues using access lists

It is also possible to protected service-controlled queues from unauthorized read, delete or write access. To do this an access list is defined (TAC/USER statement).

TAC statement in section "TAC - define the properties of transaction codes and TAC queues"
The following operands are used to control access for TAC queues.

- *tacname*

  Name of the TAC queue.

- Q-READ-ACL=

  Q-WRITE-ACL=

  Name of the access list that controls the read, delete and write access of a user to this queue. The access list must be generated using a KSET statement.

A user only has read or write access to the TAC queue if the key set of the user and the key set of the LTERM partner via which the user has signed on both contain at least one of the roles that are defined in the access list for the TAC queue.
The key set must be generated for the user and the LTERM partner using the USER or LTERM statements.

USER statement in section "USER - define a user ID"
The following operands can be used to control the access for USER queues.

- *username*

  UTM user ID.
- KSET=

  Specifies the name of the key set to which the user ID is assigned.
  The key set must be defined using the KSET statement. Each user may be assigned a maximum of one key set.
- Q-READ-ACL=

  Q-WRITE-ACL=

  Name of the access list via which the user is able to protect their own USER queues from read, delete or write access. The access list must be generated using the KSET statement.

  > The owner of a queue always has read, write and delete authorization for their queue, regardless of whether the read/write authorizations are restricted for other users.
  >
  > An external user only has read or write access to the USER queue of another user if the key set of the external user and the key set of the LTERM partner via which the external user has signed on each contain at least one of the roles defined in the access list for the USER queue.
  >
  > If you do not specify Q-READ-ACL/Q-WRITE-ACL all users have read, delete and write authorization within the queue.

For more detailed information on Message Queues see "Generating service-controlled queues".

4.9.3 Data access control with distributed processing

You can use the data access control mechanisms of openUTM with distributed processing. The protection methods are specified when the applications are generated. A distinction is made between the job-submitting and job-receiving service.

Protection methods for job-submitting services

When generating an application you generally initially specify which services of a remote partner application may be called. For each remote service that is to be used you must agree an LTAC local transaction code (LTAC statement). Access is generally denied to remote services for which no LTACs have been agreed.

In order to further graduate the data access control you can also assign lock codes to individual LTACs (see "Lock /key code concept") or use access lists (see "Access list concept").
A service of the local application can only address a remote service if the service was started under a user ID (KCBENID) and from a client (KCLOGTER) that have the appropriate access permissions.

LTAC statement in section "LTAC - define a transaction code for the partner application"
The following operands are used to define which services of a remote partner application may be called and which access authorizations are placed on the LTAC. The operands ACCESS-LIST and LOCK are mutually exclusive.

- *ltacname*

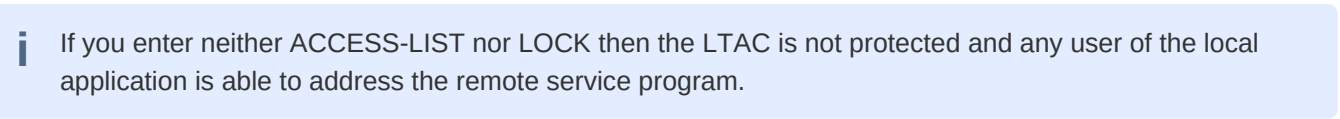
  Name of a local TACs (LTAC) for the remote service program.

- ACCESS-LIST=

  Name of an access list. In order to be able to start the remote service program the key set of the user of a local application must have been assigned at least one of the roles defined in the access list (as defined in the USER statement).
  The access list must be defined using a KSET statement.

- LOCK=

  Definition of the lock code for access to the remote service program. A service of the local application can only address this remote service if the local service was started under a user ID (KCBENID) and from a client (KCLOGTER) that have the appropriate access permissions.
  ACCESS-LIST and LOCK cannot be specified simultaneously.



Protection measures for job-receiving services

You protect job-receiving services by assigning a key set to the logical access point of a partner application (LPAP or OSI-LPAP). Only if this key set contains a key code or access code that corresponds to the lock code or access list of the requested service is it possible for the process requested by the partner application to be started.

In order to be able to access a remote service, the service that is being called must be generated with a TAC and the following conditions must be fulfilled:

- LU6.1 connections:

  The key set for the partner as defined in LPAP ...,KSET= must contain a key code that corresponds to LOCK= or ACCESS-LIST= of the TAC.

- OSI TP connections:
  - If a partner attempts to sign on without a user ID, then the key set defined in OSI-LPAP ...KSET and OSI-LPAP ...,ASS-KSET= must contain a code that correspond to LOCK= or ACCESS-LIST= of the TAC.

    The access authorizations are derived from the intersection of the key sets of KSET= and ASS-KSET=. Thus KSET= should always be a superset of ASS-KSET=.
    You can define suitable restrictions on the key set defined with OSI-LPAP ...,ASS-KSET to ensure that specific TACs cannot be called unless the partner specifies a real user ID.
  - If a partner attempts to sign on with a real user ID, then the key set of this user ID and that defined in OSI-LPAP ... KSET= must contain a code that corresponds to LOCK= or ACCESS-LIST= of the TAC.

  This also applies to a client/server link with OpenCPIC.

For more detailed information about data access control with distributed processing see openUTM manual “Concepts und Functions”.

4.10 Message encryption on connections to clients

Clients often access UTM services via open networks. This may give unauthorized persons the opportunity to read data from the line and obtain passwords for UTM user IDs or sensible user data, for example. To prevent this, openUTM supports the encryption of passwords and user data on connections to UPIC clients and on BS2000 systems additionally on connections to certain terminal emulations.

Encryption in openUTM not only serves to secure the data on the connection between the client and the server application, but it can also be used to limit access for clients and access to certain services. Up to two encryption levels are available for selection (AES-CBC or AES-GCM algorithm, see "Data access control").

When communication with USP-socket applications or HTTP clients TLS connections can be used to allow for encrypted exchange of messages between the communcsation partners. A transport system access point for TLS connections is setup with the statement BCAMAPPL ..., T-PROT=(SOCKET, ..., SECURE), see chapter "BCAMAPPL - define additional application names".

4.10.1 Requirements

Connecting a UPIC client to a server application

The requirement for encryption between an openUTM server application and a UPIC client is the availablitiy of cryptographic functions.

- In openUTM server applications on BS2000 systems the encryption functionality is always available.
- In openUTM server applications on Unix, Linux and Windows systems the encryption functionality is only available after successful loading of an appropriate openSSL library.
- As UPIC client an openUTM-Client for the UPIC carrier system must be used that supports the encryption functionality. For UPIC clients on BS2000 this is always the case. For UPIC clients on Unix, Linux and Windows systems the encryption functionality is only available after successfully loading an appropriate openSSL library.

Details on the use of the openSSL library in Unix, Linux and Windows systems can be found in the openUTM manual "Using UTM Applications on Unix, Linux and Windows Systems".

Connecting a terminal emulation to a server application (BS2000 systems only)

The encryption of VTSU is offered for connections between UTM applications on BS2000 systems and terminal emulations. VTSU-B uses a separate key management. The data and system access control mechanisms that come in conjunction with encryption are in effect, however. openUTM receives information from VTSU via the encryption level that was negotiated for the connection to the client.

The following requirements must be fulfilled:

- One requirement is the use of VTSU-B and the VTSU-SEC selectable unit. Which of the current versions you must use is described in the release notes for openUTM. You can consult the Release Notice for VTSU-SEC to determine which VTSU parameters must be set.
- A terminal emulation must be in use on the client that supports the encryption functions.

These communication partners are called VTSU partners in the following.

4.10.2 Encryption methods

openUTM uses either a combination of the algorithms RSA (named after the authors Rivest, Shamir and Adleman) and AES-CBC (Advanced Encryption Standard Cipher Block Chaining Mode), or a combination of Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) and AES-GCM (Advanced Encryption Standard Gallois Counter Mode). The second combination is more modern and offers advanced security compared to the first combination, but is currently only supported by openUTM on Unix, Linux and Windows systems.

RSA-AES-CBC cipher suite
With the combination RSA-AES-CBC, before transmission the AES key is encrypted with the public RSA key of the UTM application. In order to do so openUTM generates an RSA key pair consisting of a public and a private secret key.
For a UTM application RSA keys with a length of 1,024 and/or 2,048 bits can be generated.
BSI recommends using RSA keys with a length of at least 2,000 bits.

- The public RSA key is transferred from the UTM application to the client when the connection is established. To prevent man-in-the-middle (MiM) attacks on the communication, a user should also read out the public RSA key of the application via administration, transfer it separately to the client and enter it into the client configuration.
- For each new connection, the client generates a new 128-bit AES key, encrypts it with the public RSA key of the server, and transmits it to the UTM application. The AES key is connection-specific, i.e. a separate key is generated for each connection and this key is only used for this connection.
- The UTM application decrypts the AES key using its private RSA key.

User data and passwords are encrypted on a connection with the symmetric AES key, i.e. client and UTM application use the same AES key to encrypt and decrypt messages.

ECDHE-RSA-AES-GCM cipher suite (only for Unix, Linux and Windows systems)
The combination ECDHE-RSA-AES-GCM uses the elliptic curve based Diffie-Hellman method to generate an AES session key. Each side generates a Diffie-Hellman key pair, transmits the public part of its key pair to the partner and generates the common AES session key with its private key and the public key of the partner. This means that in this procedure the AES session key is not transferred on the data connection.

The server also signs its public Diffie-Hellman key with the private RSA key of the UTM application. In this way, the client can verify that the Diffie-Hellman public key sent from the server really belongs to the UTM application. Again, as described above, to defend against man-in-the-middle attacks, the public RSA key should be made known to the client separately.

The Ephemeral Diffie-Hellman method offers the user **Perfect Forward Secrecy**; this means that even recorded data cannot be decrypted later if the long-term key (RSA key) should later be cracked.

The AES-GCM algorithm is used to encrypt user data. One of the advantages of this method over AES-CBC is that it supports **Authenticated Encryption with Associated Data** (AEAD), in which the encrypted user message and the other protocol parts of the message are protected against changes by a Message Authentication Code (MAC).

Passwords are encrypted with AES-CBC as described above.

4.10.3 Encrypting passwords and user data

User data and passwords are not passed in encrypted form on connections between UTM application and trusted clients (i.e. clients generated trusted clients; see point 3 in chapter "System access control").

Passwords from (non-trusted) UPIC clients are always encrypted and then passed to the UTM application in openUTM if the client as well as the server supports encryption. Passwords are also encrypted in this case if no encryption was agreed to for the connection.

BS2000 systems

Passwords are only passed in encrypted form on connections between UTM applications on BS2000 systems and VTSU partners if encryption was agreed to for the connection or if the password was entered in a blanked-out field.

The encryption of **user data** is optional. This is negotiated between the client and the server when a UPIC conversation or connection to a VTSU partner is established.

- The client can force encryption.
  The ENCRYPTION-LEVEL keyword in the Side Information file and the *Set_Encryption_Level* function call are available for a UPIC client for this purpose.

  BS2000 systems

  The encryption level is defined on the host for VTSU partners. Various encryption levels can be specified, from unconditional encryption for all applications through the encryption of individual messages that the user himself has selected.

- A UTM application can request encryption for a certain service or a certain partner.

If one of the partners requests encryption, then the request for encryption is either accepted by the other side or the conversation/connection between the partners is not established.

Encryption is always negotiated on a conversation-to-conversation or connection-to-connection basis. Message-specific encryption via the program interface is not possible.

You can assign every client and every service an encryption level in the configuration of the UTM application. The encryption level specified whether or not messages from the client must be encrypted. The encryption levels are defined with the KDCDEF option ENCRYPTION-LEVEL in the TAC, PTERM and TPOOL statements.

The encryption levels can be used by openUTM to control the access of clients as well as the access to certain services.

4.10.3.1 System access control

You can specify an encryption level for every client (PTERM) and every client group (LTERM pool; TPOOL) in the UTM configuration. The encryption level specifies if and how clients must encrypt messages or may encrypt messages. In this manner a UTM application can protect itself from accesses via insecure clients.

You specify the encryption level for a client in the KDCDEF generation in the PTERM or TPOOL statement of the client:

```
PTERM ...,ENCRYPTION-LEVEL=
```

```
TPOOL ...,ENCRYPTION-LEVEL=
```

There are following variants:

1. openUTM requests the use of encryption from the client.
   The client must encrypt in all cases, otherwise it will not gain access to the UTM application. The minimum length of the RSA key used is predefined. If the partner does not support encryption or cannot use the RSA key of the requisite key length, then it cannot establish any connections to the UTM application.
   In this case, generate one of the following variants:

   `ENCRYPTION-LEVEL=3` (RSA key length 1024 bit, AES-CBC method)
   `ENCRYPTION-LEVEL=4` (RSA key length 2048 bit, AES-CBC method)
   For LUW platforms additionally:
   `ENCRYPTION-LEVEL=5` (RSA key length 2048 bit, ECDHE-RSA-AES-GCM methods)
2. openUTM does not request encryption and the client can specify whether or not the connection is to use encryption.
   The client is also allowed access without encryption, but it must encrypt if a service explicitly demands it (see "Data access control").
   In this case, generate:

   `ENCRYPTION-LEVEL=NONE`
3. The client is trusted (*trusted client*).
   Encryption is not used on connections to such clients. A trusted client can also call „protected“ services without encryption (see section below).
   You should only generate clients as trusted when you are sure that communication occurs via a secure line.
   In this case, generate with:

   `ENCRYPTION-LEVEL=TRUSTED`

> ¡ Socket and HTTP clients which connect to the UTM application via a secure connection are always trusted clients (see statement BCAMAPPL T-PROT=(SOCKET,..., SECURE))
>
> *Unix, Linux and Windows systems:*
> Local UPIC clients (type UPIC-L) are always trusted clients.

4.10.3.2 Data access control

You can protect individual services from accesses via insecure clients with the help of the encryption functions. A client may only access protected services if it is a trusted client or if it is able to encrypt using the requisite method.

You can protect a service by assigning encryption level 2 or 5 to the corresponding service TAC´(5 only on Unix, Linux and Windows systems):

`TAC ...,ENCRYPTION-LEVEL=2` (encryption according to the AES-CBC algorithm)

`TAC ...,ENCRYPTION-LEVEL=5` (encryption according to the AES-GCM algorithm)

If a service is protected in this manner, then the following is true:

- A client generated as trusted can start such a service without using encryption.
- For non-trusted clients the service belonging to the transaction code is only started if the client has passed the input message encrypted with the requisite method. Otherwise
  - In the case of UPIC clients, conversation establishment is rejected by openUTM.
  - In the case of VTSU partners, this leads to a BADTAC or message K009 is output.

  If the service is called via a transaction code without user data (e.g. for terminal emulations via a function key) or started due to service chaining, then the service is also started without encryption. openUTM encrypts then all dialog output messages to the client. openUTM expects all further input messages from the client to be encrypted for multi-step services. If the input message contains unencrypted user data, then the service is terminated abnormally.

Encryption is optional when you generate a service TAC as follows (default):

`TAC ...,ENCRYPTION-LEVEL=NONE`

Information for encryption on the KDCS program interface

You also have the possibility of writing separate program units that execute an access authorization check. Encryption data is displayed on the program interface for the INIT PU call. The following information is displayed:

- the encryption levels that are generated for the client and transaction code
- whether encryption was negotiated for the conversation
- whether the client supports encryption in principle
- whether the last input message was encrypted

4.10.4 Creating the RSA key pair and reading the public key

You should replace the RSA key pair with a new RSA key pair in your UTM application in regular intervals for security reasons. This is especially important if you use encryption levels less than 5. The administration program interface and the administration tools WinAdmin and WebAdmin provide the corresponding functions.

See the openUTM manual “Administering Applications”; KDCADMI operation code 4KC_ENCRYPT or the online help system for WinAdmin or WebAdmin, keyword „RSA keys“.

With the help of the administration you can create a new key pair, read the public key and activate the new key pair. Only after activation can the new key pair be used by the UTM application for encryption. An activated key pair can also be deleted using administration facilities.

To further increase the security of the data on a connection you should read the public key of the RSA key pair, pass it to the client using your own method and store it there. You should only activate the new RSA key pair once this has been accomplished. With the help of the public RSA key you have stored, the client can verify if the public key received over the connection to the UTM application really came from the UTM application.

4.11 Defining database linking

When configuring the application you must use the KDCDEF control statements to define the database system with which the UTM application is to coordinate.

> If a UTM application is to be linked with a database, additional parameters must be specified when linking and starting. See also the openUTM manual “Using UTM Applications”.
> The remaining UTM generation is not affected by the linking.

4.11.1 Linking databases on BS2000 systems

openUTM supports coordination with the following database systems:

- UDS/SQL
- SESAM/SQL
- XA
- CIS
- LEASY (the LEASY file systems behaves like a database system in relation to openUTM)

A UTM application can work in coordination with up to three (up to eight with special release) different databases. Each database system is defined with a DATABASE statement for the KDCDEF run.

DATABASE statement in section "DATABASE - define the database system (BS2000 systems)": Definition of the database with which the UTM application works together:

- ENTRY=

  Entry name of the supported database, which can be seen in the table in section "DATABASE - define the database system (BS2000 systems)".
- USERID= and PASSWORD=

  User name and password for the database system, supported only for Oracle databases (TYPE=XA).

  > Alternatively, the user name and the password can be transferred to the database system by means of start parameters.
  > It is possible to modify the user name and/or the password by means of dynamic administration.

- LIB=

  Object module library from which the connection module to the database system is to be loaded dynamically.
- TYPE=

  Type identifier of the database system.

  - You can connect to database systems not contained in the list above but that support the IUTMDB interface with TYPE=DB.
  - The link to a XA resource is generated with TYPE=XA.

4.11.2 Linking to a Resource Manager on Unix, Linux and Windows systems

openUTM is linked with Resource Managers (e.g. database systems) via the XA interface standardized by X/Open. It coordinates the transactions of openUTM with the services of the Resource Manager. The XA interface is supported in the CAE version of the XA interface (XA-CAE).

openUTM for Unix, Linux and Windows systems supports coordination with the data base system Oracle.

RMXA statement in section "RMXA - define a name for an XA (database) connection (Unix, Linux and Windows systems)"
The Resource Manager to which openUTM is to be linked and the version of the XA interface via which the link is to be made must be defined in the generation with the RMXA statement:

- XASWITCH=

  Name of the xa_switch_t structure of the Resource Manager, which is made known to openUTM.
- USERID= and PASSWORD=

  User name and password for the database system, supported only for Oracle databases.

  > Alternatively, the user name and the password can be transferred to the database system by means of start parameters.
  >
  > It is possible to modify the user name and/or the password by means of dynamic administration.

- XA-INST=

  Name of the XA instance.

The following must be noted for the linked operation of openUTM with XA:

- Several Resource Managers (i.e. database systems) can be served within a UTM application.
- When multiple instances are to be generated, these must be defined with the same xa_switch and different instance names (XA-INST operand).
- The simultaneous operation of several entities (databases) of a Resource Manager (database system) is possible provided the Resource Manager supports multi-instance mode. The databases with which the UTM application is linked are determined by corresponding start parameters for the application. For multi-instance mode, you must define several RMXA statements and specify several open strings at the start.

Below is a description of how you must generate the linking of your UTM application with the Resource Manager. The database-specific names specified here (xa_switch_t structure) may change, which is why you should check that the specifications are correct. For more information, see the documentation for the database systems.

Linking with Oracle

```
RMXA XASWITCH=xaosw,USERID= ...
```

or

```
RMXA XASWITCH=xaoswd,USERID= ...
```

Detailed examples are provided in openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems” under the heading "Start parameters for a UTM database application".

When the dynamic XA connection is used on Windows systems, the link between Oracle and openUTM must be configured in addition in the Windows Registry. Details siehe openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”, Stichwort „UTM-Datenbank-Anschluss generieren“.

4.12 Internationalizing the application - XHCS support (BS2000 systems)

A UTM application on BS2000 system can be programmed such that communication partners with different languages can receive the messages from the program units in their respective language. Even regional differences within a language can be taken into account. Date specifications, time, units of measurement, and currency symbols can be displayed in accordance with language-specific conventions.

To display the fonts and special characters of the individual languages on a terminal or printer, you may require various extended character sets (8-bit codes or Unicode). Using the BS2000 software product XHCS (E**x**tended **H**ost **C**ode **S**upport), several extended character sets can be used simultaneously on a BS2000 system. openUTM supports the functions of XHCS. This means that you can assign a particular language environment – also called a locale – to the UTM objects. In other words, you can assign a standard locale. Individual users and LTERM partners that clients use to connect to the application are assigned specific locales that are used to edit the messages.

To implement multilingualism in UTM applications, openUTM offers the following functions.

- When generating the application, specific languages and the character sets to be used for output can be assigned to the application, the user IDs, the LTERM partners, and the LTERM partners of the LTERM pools. In this case you define locales, which define the language environment and character set, for the objects.
- You can define locales for user message modules that take account of language-specific requirements. These language-specific message modules are assigned to users and LTERM partners whose language and territorial identifiers match the locale of the language-specific message module. See also chapter "UTM messages".
- While the application is running, you can change the assignment of language and character set for your user ID. The KDCS interface provides the SIGN CL call for this purpose.
- Using the variants INIT PU and INFO LO of the function calls INIT and INFO, a UTM program can read the language and character set of the user ID, the application, a particular LTERM partner, or the LTERM partners in a pool. The program unit thus obtains information on the character sets supported by the terminal and the character set of the input message. With this information, the program unit can correctly interpret the input of the user and send messages to the user in the correct language and with the appropriate character set.
- If the message of a program unit is sent to a terminal/printer, openUTM transfers the logical message of the program unit to VTSU-B together with the name of the character set to be used for editing. VTSU-B edits the message for outputting to the terminal or printer. For information on the character set is used to edit a message please refer to the openUTM manual „Programming Applications with KDCS”.

  If the job submitter in a service is a partner program, the logical message is transferred to the job submitter without editing.
- The program unit can use INFO LO to request information from openUTM regarding the language and character set of the LTERM partner, and the character sets supported by the terminal/printer assigned to this LTERM partner. The character set used to edit the message for outputting to the terminal/printer must be compatible with one of the character sets supported by this terminal/printer.

Before discussing these functions, we will explain specific XHCS terms.

4.12.1 Definitions of XHCS terms

ISO character sets, variant numbers

Various extended character sets for various language areas are standardized in ISO 8859, for example ISO 8859-1, ISO 8859-2, etc. The numbers at the end (-1, -2, etc.) are called the *variant numbers*. An extended character set contains all the characters required to represent the language of a language area.
ISO 8859 codes are extensions of the ASCII code ISO 646. They are used by terminals and Unix, Linux or Windows systems, for example. All ISO 8859 character sets contain the ASCII code as the shared part in the low-order half of the code table.

In addition there are the Unicode character sets UTF-8 and UTF-16 with which the characters of all language areas can be represented with a single character set.

EBCDIC character sets

EBCDIC character sets are used in the BS2000 operating system. An extension of EBCDIC.DF.03-IRV or -DRV exists for each ISO 8859 code. EBCDIC.DF.03-IRV is the international reference version and EBCDIC.DF.03-DRV is the German reference version of the non-extended EBCDIC code. Both codes contain the EBCDIC kernel as the shared character set and only differ in certain symbols. The extensions of these EBCDIC character sets are called EBCDIC.DF.04-1, EBCDIC.DF.04-2 through EBCDIC.DF.04-F.

The EBCDIC counterpart of ther Unicode character sets UTF-8 and UTF-16 is the character set UTF-EBCDIC (UTFE).

Compatible character sets

Extended ISO and EBCDIC character sets with the same variant number are *compatible*, i.e. they contain the same characters. The individual characters are located at different code positions within the code table. The codes can be transferred using conversion tables.

The BS2000 system administrator can use XHCS to modify the EBCDIC character sets by assigning different code positions in the code table to the individual characters of a character set. The complete set of characters is retained. The modified EBCDIC character sets are *compatible* with the EBCDIC.DF.04-n character set from which they were generated.

Reference code

XHCS combines all compatible character sets of the system into a group. A group therefore contains an ISO variant and the EBCDIC character sets compatible with this variant. The EBCDIC.DF.04-n character set of the group is the reference code of the group. All character sets in a group can be converted to the reference code of the group using XHCS.

Coded character set name (CCS name)

A name containing a maximum of eight characters, known as the CCS name or the CCSN, is assigned to each character set used in the system. The CCS name uniquely identifies the character set in the system. The CCS names of the reference codes are predefined by XHCS. EBCIDIC.DF.04-1 has the CCS name EDF041, for example.

A list of the CCS names for the character sets available in your BS2000 can be obtained using EDT. To request this information, call EDT and enter the EDT statement `@SHOW CCS`. EDT then supplies a list of the available character sets.

Default system code

The BS2000 system administrator can define several extended character sets (also for various ISO variants), which can be used simultaneously by the system components.

The system administrator can define one of these character sets as the default system code. The default system code currently set is indicated in the output of the command/SHOW-SYSTEM-PARAMETERS PAR=*AL. It is specified in the HOSTCODE parameter.

Default user character set

The BS2000 system administrator can assign one of the character sets defined in the system as the default user character set for each BS2000 user ID. If a default user character set is defined for the BS2000 user ID, its CCS name is displayed in the output field CODED-CHARACTER-SET of the /SHOW-USER-ATTRIBUTES command.

> For further information on XHCS, see the User Guide “XHCS 8-Bit Code Processing in BS2000/OSD - Internationalization”.

4.12.2 Defining the language environment - setting the locale

When generating a UTM application, a separate language environment can be defined for the UTM application, for each LTERM partner, for all LTERM partners in an LTERM pool, and for each user ID. To do this, you assign the application and the individual objects a triplet comprising the language identifier, territorial identifier, and name of a character set, which is known as the locale. The locale is specified as follows:

```
LOCALE=( lang_id,terr_id,ccsname )
```

lang_id The language identifier *lang_id* identifies the language in which the user is to be addressed by the UTM program units. The language identifier can be up to 2 bytes long. The descriptor of a language can be freely selected.

terr_id The territorial identifier *terr_id* enables you to take account of regional differences within a language (e. g. English in UK or America) or different units of currency and measurement in the various countries (dollar and sterling). The territorial identifier can be up to 2 bytes long and can be freely selected.

ccsname The character set name *ccsname* specifies which character set can be used to edit a message for outputting to the terminal. As the character set name, specify the CCS name of a character set defined in the BS2000 system. CCS names are assigned by the BS2000 system administrator.

If all users come from the same language area, e.g. Western Europe, it is sufficient to assign an extended character set to the UTM application. It is only necessary to use user specific character sets if the various users of an application speak languages that cannot all be represented by an extended character set.

In order to support extended character sets, the subsystem XHCS must be available on the processor on which the UTM application is running. For all character set names generated in the UTM application, associated EBCDIC character sets must be defined in XHCS. In addition, the terminals must support an ISO character compatible with the respective EBCDIC character set. Only particular types of terminals and printers support 8-bit character sets.

Application-specific language environment – standard-language environment

MAX statement in section "MAX - define UTM application parameters"
You assign the locale to theUTM application in the generation using the MAX statement:

- LOCALE=

  The locale generated for the application is assigned to each user ID, each LTERM partner, and each LTERM pool as the default value for the language environment. This default setting applies as long as no specific locale is defined for these objects.

User-specific language environment

USER statement in section "USER - define a user ID"
You assign a locale to a user ID using the USER statement:

- LOCALE=

  The character set assigned to a user ID is used to output dialog messages to the screen (see the character sets section of the openUTM manual „Programming Applications with KDCS”).

LTERM partner-specific language environment

LTERM statement in section "LTERM - define an LTERM partner for a client or printer" and **TPOOL statement** in section "TPOOL - define an LTERM pool"
You use the LTERM statement to assign a locale to an LTERM partner via which a terminal or printer connects to the application. For an LTERM pool, a locale is defined for all LTERM partners in this pool using the TPOOL statement:

- LOCALE=

  The character set defined for the LTERM partner is used to output asynchronous messages (see also "UTM messages").

  The LTERM partner-specific locale is also used in the first part of the sign-on service, for example, if the user has not yet signed on, i.e. the user-specific language environment is not yet created.

Example

The language identifier DE for German is used in the application. To be able to take account of the different units of currency in messages to users in Germany and Switzerland (Euro and franc), the territorial identifiers `De` for Germany and `CH` for Switzerland are defined. The EBCDIC character set EBCDIC.DF.04-1 can be used to output messages. Its CCS name is EDF041.

- The locale for users in Germany can be defined as the standard language environment for the application. To this end, specify the following in the MAX statement:

  ```
  MAX ..., LOCALE=(DE,DE,EDF041)
  ```

  In this case, no separate locale need be defined for users and terminals in Germany that connect to the application via LTERM partners.

- If language-specific requirements are to be taken into account for users and terminals in Switzerland, the following must be generated:

  ```
  USER username ,..., LOCALE=(DE,CH,EDF041)
  LTERM ltermname ,..., LOCALE=(DE,CH,EDF041)
  ```

- However, you can also use the DEFAULT statement to set the locale (DE,DE,EDF041) for all USER and LTERM statements:

  ```
  DEFAULT USER LOCALE=(DE,DE,EDF041)
  DEFAULT LTERM LOCALE=(DE,DE,EDF041)
  ```

  You must then also generate the following for users and terminals in Switzerland that connect to the application via LTERM partners:

  ```
  USER username ,..., LOCALE=(,CH)
  LTERM ltermname ,..., LOCALE=(,CH)
  ```

4.12.3 Character set names for edit profiles and formats

In addition to the user-specific and LTERM partner-specific assignment of character set names, a separate character set name can be assigned to each edit profile defined in the application.

The name of a character set can be assigned to each format when creating formats with FHS/IFG.

For information on which of the generated character set names (application-specific, user-specific, LTERM partner-specific character set, or the character set name assigned to an edit profile or to a format) is used to edit a message for outputting to the screen or printer, as described in section "Character sets for editing messages".

4.12.4 Querying the language environment in a UTM program unit

In the initialization phase, openUTM transfers information to a program unit regarding the locale of the user who started the associated service. The prerequisite is that the INIT call is used with the operation modification PU and that the program requests the information. See also the openUTM manual „Programming Applications with KDCS”.

If the user has not yet signed on, openUTM transfers the locale of the LTERM partner via which the connection to the application was established. The program unit can then correctly interpret the code and terminology in the input from the communication partner and can generate messages in the language used by the communication partner.

Specifications on the user-specific character set are required because the user-specific character set is used to output dialog messages to 8-bit terminals if no edit profile or format with CCS names is assigned with MPUT. The program unit must take this into account when structuring the message.

The character set of the LTERM partner is used to output asynchronous messages to 8-bit terminals if no edit profile or format with CCS name is specified with FPUT. Information on the character sets supported by the terminal can be obtained using the KDCS call INFO LO.

In addition to the locale of the LTERM partner associated with the service, you can also use INFO LO to query the ISO character sets supported by the terminal. If the user-specific or LTERM partner-specific character set is not compatible with one of the ISO character sets supported, the service may be aborted or, in the case of asynchronous messages, the message may be lost.

4.12.5 Character sets for editing messages

If the message of a program unit is sent to a terminal or printer, openUTM transfers the logical message and a character set name to VTSU-B. VTSU-B edits the message for output. The type of message and type of client determine which of the generated character sets is transferred from openUTM to VTSU-B.

A distinction is made here between three message types:

- message in line mode without edit profile or messages created by event exit FORMAT
- message in line mode with edit profiles
- message in format mode.

Message in line mode without edit profile and messages created by event EXIT FORMAT

- The terminal or printer to which the message is addressed supports none of the extended character sets.

  The message is transferred to VTSU-B without specifying a character set name.Characters that do not belong to the EBCDIC kernel are replaced by smudge characters or substitute characters.

- The terminal or printer to which the message is addressed supports extended character sets.

  Dialog messages are edited using the user-specific character set.

  Asynchronous messages are edited using the LTERM partner-specific character set of the LTERM partner whose message queue contains the message.

  In this way, the program unit can correctly serve 8-bit printers, amongst others. The program unit uses INFO LO to obtain information on the character set names generated in the locale of the LTERM partner assigned to the printer. It then transfers the character set name together with the message to VTSU for editing.

  You must ensure that the EBCDIC character set associated with the character set name is compatible with an ISO character set supported by the printer. This is not checked by VTSU-B, as VTSU-B does not know which ISO character sets are supported by the printer. The supported character sets are indicated in the printer description.

  In order that VTSU-B can serve the printer in 8-bit mode, the VTSU-operating parameters xxxxxDEV8 and xxxxxLIN8 must be set. Any modification to the operating parameters does not come into effect until VTSU is loaded dynamically. See the User Guide “VTSU - Virtual Terminal Support”.

  If the EBCDIC character set of the message is not compatible with any ISO character set of the 8-bit client, one of the following errors will occur:

  - A dialog service is terminated with PEND ER. A PEND ER dump is created and UTM message K017 is sent.
  - An asynchronous output message is discarded. A UTM dump is created and a UTM message is sent.
  - UTM message K106 is output to the screen if an asynchronous message was retrieved with KDCOUT (LTERM...,ANNOAMSG=YES) or if part of the message was already sent before the error occurred.
  - An error message is written to SYSLOG.

  In relation to the character set, formatted messages created by the format exit are handled like messages in line mode.

Message in line mode with edit profiles

- **No** character set name is assigned to the edit profile.

  In relation to the character set used, these messages are handled like messages in line mode without edit profile.

  By using user-specific and LTERM partner-specific character sets, you can thus permanently serve users in 8-bit mode without having to explicitly generate 8-bit edit profiles.

- A character set name is assigned to the edit profile.

  openUTM always transfers the character set name of the edit profile together with the logical message to VTSU-B for editing.

  If a message is sent to a terminal or printer that only permits 7-bit mode, the service is terminated with PEND ER in the case of dialog services. An asynchronous message is discarded in this case and a UTM message is sent.

  In the case of messages to 8-bit terminals or printers, the EBCDIC character set used must be compatible with an ISO character set supported by the terminal/printer. With asynchronous messages, VTSU-B cannot check this compatibility. For this reason, the message should only be sent in the character set assigned to the edit profile.

  You need only explicitly assign a character set to an edit profile if this character set is not identical to the user-specific or LTERM partner-specific character set and if the message to be sent contains characters from this character set which do not, however, belong to the EBCDIC kernel.

Message in format mode

- Format to which no character set was assigned when creating IFG.

  In relation to the character set used, openUTM handles these messages in the same way as messages in line mode without edit profile, i.e. a dialog message is edited using the user-specific character set, and an asynchronous message is edited using the LTERM partner-specific character set, if this message is directed to an 8-bit terminal or corresponding printer.

- Format with character set name, i.e. a character set was assigned to the format when creating with IFG.

  The character set name assigned to the format is transferred by openUTM for editing.

  If the message is directed to printers or clients that only permit 7-bit mode, the service is terminated with PEND ER in dialog services. An asynchronous output message is discarded in this case and a UTM message is sent.

  With messages to 8-bit terminals or printers, the EBCDIC character set used by the message must be compatible with one of the ISO character sets supported by the terminal/printer.

  A character set need therefore only be explicitly assigned to the format to be sent if it contains characters that do not belong to the EBCDIC kernel.

  A character set need only be explicitly assigned to the format if the character set used for representation is not identical to the user-specific or LTERM partner-specific character set.

5 Notes on generating a UTM cluster application on Unix, Linux and Windows systems

Unlike a standalone application, a UTM cluster application is intended to be run on more than one computer. Together, these computers are known as a cluster and the individual computers on which the application is to run are known as nodes. A UTM cluster application is made up of several identically generated UTM applications (the node applications) that run on the individual nodes.

The configuration of the application, including the KDCFILEs for all nodes, is created in a single generation run and is therefore always the same.

A UTM cluster application can be distributed across up to 32 nodes.

The computers that belong to a cluster must be equivalent in terms of hardware status and software configuration. Discrepancies involving compatible correction statuses and updates are possible. Mixed configurations, such as Windows and Unix computers in combination are not possible.

> **CAUTION!**
> The nodes of a cluster must always have the same system time.

You can find detailed information on operation and in particular on generating applications for UTM cluster applications in the relevant openUTM manual “Using UTM Applications on Unix, Linux and Windows systems”.

5.1 Generating a UTM cluster application

The generation of a UTM cluster application differs in the following ways from that of a standalone UTM application:

- There are the additional statements CLUSTER and CLUSTER-NODE as well as the operand value GEN=CLUSTER in the OPTION statement, see "KDCDEF statements".
- When a UTM cluster application is generated, UTM cluster files are also generated, see below.
- Only one copy of the KDCFILE is permitted, i.e. KDCFILE=(...,SINGLE) must be specified in the MAX statement (default value).
- The size of a UTM page must be 4K or 8K (MAX statement, BLKSIZE operand)

Please also note the following important differences that apply during a UTM cluster application run:

- In a UTM cluster applications, the user data that applies globally throughout the cluster is stored in GSSB and ULS areas. In the case of UTM-F, the service data is also stored in these areas.
- The KDCFILEs of the node applications contain only the user data that is local to the node.

5.1.1 UTM cluster files

A UTM cluster application is generated in a generation run during which the KDCDEF utility creates the following files:

- the cluster configuration file
- the cluster user file
- the cluster page pool files
- the cluster GSSB file
- the cluster ULS file
- an initial KDCFILE
- and the root source

The initial KDCFILE must be copied for each node application after the generation run.

The UTM cluster files generated by KDCDEF do not have to be generated as often for a UTM cluster application as the KDCFILE or the root source.

Subsequent generation runs can be performed in order to

- modify the KDCFILE and/or the root source.

  If you modify only the KDCFILE (but not the UTM cluster files) in a subsequent generation run, please note that:

  - The sequence of TAC statements must not be modified. Otherwise services may be terminated abnormally on service restarts. As a result, you must append new TAC statements at the end and must not delete any TAC statements.
  - The RESTART parameter in the USER statements must not be modified.

  A KDCUPD run for the node applications enables you to transfer the data from the previous node KDCFILEs to the newly generated KDCFILEs, see "Update generation for UTM cluster applications".
- regenerate the UTM cluster files.

  You can perform a KDCUPD run for the UTM cluster application in order to take over the data from the previous UTM cluster files into the newly generated files, see "Update generation for UTM cluster applications".

If changes are made to the configuration, a new initial KDCFILE can, for instance, be created with additional objects in a subsequent generation run.

The initial KDCFILE must be copied for each node application after the generation run.

> It must be possible to access the UTM cluster files and the KDCFILEs of all node applications from all node applications. See also the section "UTM cluster application" in the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

Figure 17 shows what files are created when you define a UTM cluster application.

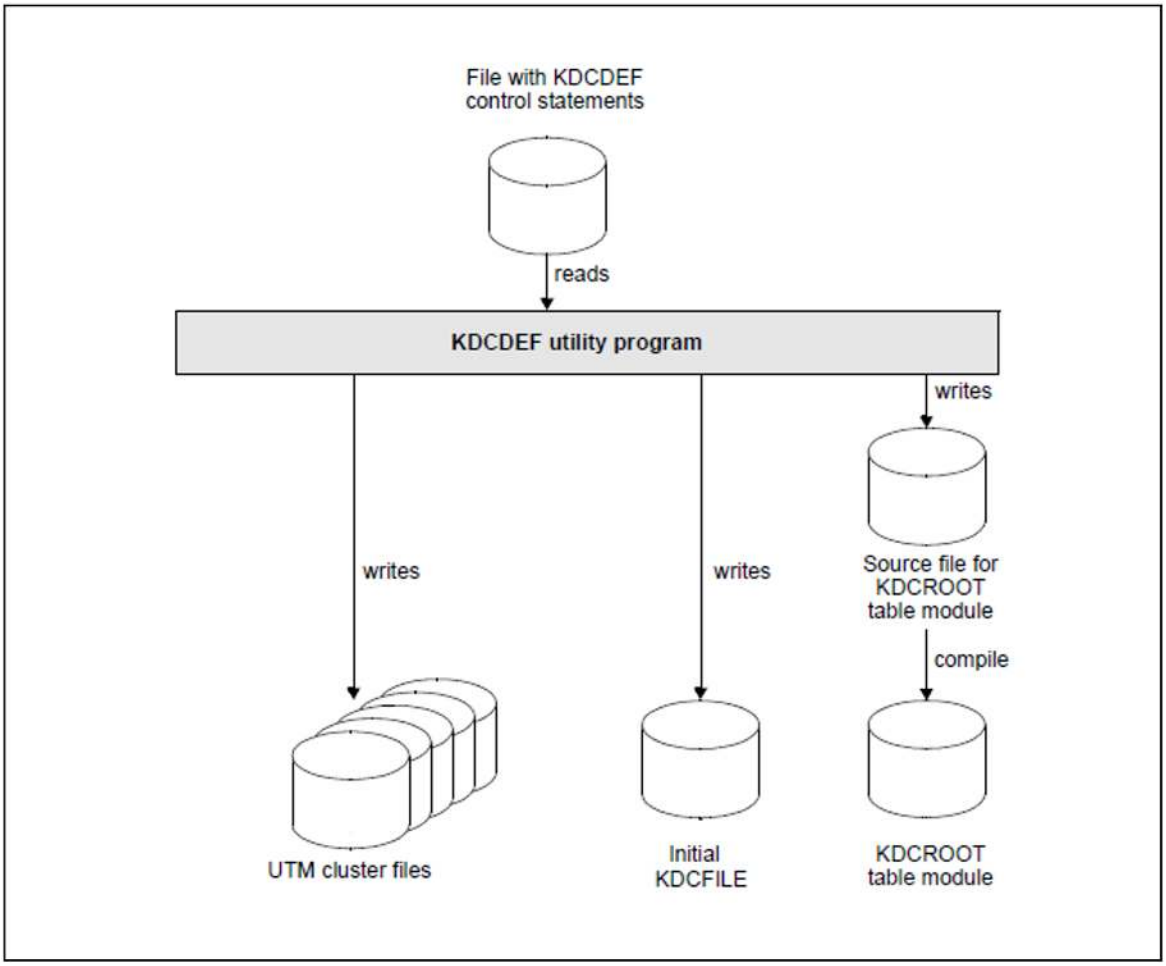


Figure 17: The result of the KDCDEF run (with OPTION ...,GEN=(KDCFILE, ROOTSRC, CLUSTER)) for a UTM cluster application.

If you also specify GEN=CLUSTER with the OPTION statement, a cluster configuration file is created together with the following files.

- Cluster user file for managing user IDs in a UTM cluster application.
- Cluster page pool files for storing user data that applies globally in the cluster in a UTM cluster application and for managing the cluster page pool.
- Cluster GSSB file and cluster ULS file for managing GSSB and ULS in a UTM cluster application.

If you specify OPTION GEN=CLUSTER, then you must also specify a CLUSTER statement and at least two CLUSTER-NODE statements.

Shared properties of the UTM cluster files

The UTM cluster files are generally only created once for a UTM cluster application. You can only use a new cluster configuration after all the node applications of a UTM cluster application have been terminated.

The UTM cluster files are created with the file name `UTM-C.` *suffix* in the directory defined with *cluster_filebase*. The UTM cluster files can be copied to a different directory in order to operate the UTM cluster application.

Cluster configuration file

The cluster configuration file contains information on all node applications of a UTM cluster application and specifications on data that is global to the cluster. It is used jointly by all node applications of a UTM cluster application.

KDCDEF creates the cluster configuration file with the suffix **CFG**. The full file name or path name is:

- Unix and Linux systems: *cluster_filebase*/UTM-C.CFG
- Windows systems: *cluster_filebase*\UTM-C.CFG

Cluster user file

The cluster user file is used for managing users in a UTM cluster application.

The cluster user file can be extended during operation of a UTM cluster application. This always happens when the administrator defines new users for a UTM cluster application. You must therefore always also specify the cluster user file during subsequent generation runs for creating a new KDCFILE. The entries in the new KDCFILE are merged with the entries in the existing cluster user file and where necessary, KDCDEF extends the cluster user file to include entries for new users.

KDCDEF creates the cluster user file with the suffix **USER**. The full file name or path name is:

- Unix and Linux systems: *cluster_filebase*/UTM-C.USER
- Windows systems: *cluster_filebase*\UTM-C.USER

Cluster page pool files

The cluster page pool files are used to record user data that is managed for the entire cluster in a UTM cluster application. This data consists of the GSSBs, ULS and the user service data. The number of cluster page pool files is defined at generation time. Between one and a maximum of ten files can be created.

KDCDEF creates the cluster page pool files with the with the suffix CP*nn* where *nn*= 01, 02 up to a maximum of 10. The full file name or path name of a cluster page pool file is:

- Unix and Linux systems: *cluster_filebase*/UTM-C.CP *nn*
- Windows systems: *cluster_filebase*\UTM-C.CP *nn*

A control file for the cluster page pool is also always created. The name of this file includes the specification UTM-C.CPMD.

Cluster GSSB file

The cluster GSSB file is used for managing GSSBs in a cluster application.

KDCDEF creates the cluster GSSB file with the suffix **GSSB**. The full file name or path name is:

- Unix and Linux systems: *cluster_filebase*/UTM-C.GSSB
- Windows systems: *cluster_filebase*\UTM-C.GSSB

The cluster GSSB file can be extended while an application is running. This is done whenever the space left in the file is no longer sufficient to accept the current management information.

Cluster ULS file

The cluster ULS file is used for managing ULSs in a cluster application.

KDCDEF creates the cluster ULS file with the suffix **ULS**. The full file name or path name is

- Unix and Linux systems: *cluster_filebase*/`UTM-C.ULS`
- Windows systems: *cluster_filebase*\`UTM-C.ULS`

The cluster ULS file can be extended while an application is running. This is done whenever the space left in the file is no longer sufficient to accept the current management information.

5.1.2 KDCDEF statements

Special generation statements are required for generating a UTM cluster application:

- You define global properties of a UTM cluster application using the CLUSTER statement. See "CLUSTER - Define global properties of a UTM cluster application". These include, for instance
  - the cluster filebase
  - the BCAMAPPL name for cluster-internal communication
  - timers for monitoring
  - a failure command and an emergency command to be called if a node fails
  - specifications on the cluster page pool files (number, warning level, size)
  - specifications concerning behavior on user sign-on as well as on deadlock handling.
- You specify node-specific properties for each node application with the CLUSTER-NODE statement. See "CLUSTER-NODE - Define a node application of a UTM cluster application". These include, for instance
  - the base name of the KDCFILE, the user log file and the system log file SYSLOG
  - the host name of the node
  - the reference name of the node application

  You must issue a separate CLUSTER-NODE statement for each node application.

i
- If specifications in the CLUSTER statement or CLUSTER-NODE statements are modified then it is always necessary to create a complete, new generation. This means that the KDCFILE and the cluster files must be regenerated. The only exception are increases in the size of the values for the cluster page pool. For details, see information on enlarging the cluster page pool in the relevant openUTM manual “Using UTM Applications”.
- If the UTM cluster files are to be created on generation, you must specify the GEN=CLUSTER parameter in the OPTION statement (see also "OPTION - manage the KDCDEF run").
- You must specify MAX BLKSIZE=4K or 8K during KDCDEF generation for UTM cluster applications. For applications on 32-bit systems, the default value is 4K. On 64-bit systems, the default value is 8K.
- It is not possible to generate a UTM cluster application with two copies of the KDCFILE, i.e. the value MAX KDCFILE=(..., SINGLE) must be specified (this is the default value).

5.1.3 Initial KDCFILE

In the same way as with a standalone application, the initial KDCFILE is stored under the base name that you specify in the KDCFILE operand of the MAX statement.

Each node application uses a copy of the initial KDCFILE at runtime of a node application. To allow this, you must copy the initial KDCFILE once for each node application after the generation run.

Because each node application is monitored by another ode application, all node applications must have mutual access to all KDCFILEs.

You will find information on starting and monitoring the node applications and detecting failures in the relevant openUTM manual “Using UTM Applications”.

5.2 Generating a reserve node application

During generation with KDCDEF, you have the option of creating reserve node applications with provisional values. You can subsequently use the administration facilities to change the host name and the base name of the KDCFILE of these node applications. The node application must not be active when this is done.

> To do this, specify the provisional, node-specific base name of the KDCFILE and the host name of the reserve node using the CLUSTER-NODE statement. See "CLUSTER-NODE - Define a node application of a UTM cluster application".

> At a subsequent time, you use KC_MODIFY_OBJECT administration statement to change the node-specific properties of the reserve node application: Specify the object type KC_CLUSTER_NODE to assign the spare node application actual values for the host name of the cluster node and the base name of the KDCFILE of the node application.

For further details on possibilities for using reserve node applications, refer to the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

For detailed information on changing the node-specific properties using the administration facilities, refer to openUTM manual “Administering Applications”.

5.3 Using global memory areas

GSSB and ULS

The UTM GSSB and ULS memory areas are available throughout the cluster in UTM cluster applications. This means that all node applications have read and write access to these areas. The user data is stored in cluster page pool files (see "UTM cluster files") and the management data for the GSSB and ULS areas is stored in the cluster GSSB file and cluster ULS file, respectively, see "UTM cluster files".

You can use the KDCDEF statement CLUSTER ... DEADLOCK-PREVENTION to specify whether or not openUTM is to perform additional checks to prevent deadlocks if memory areas are locked.

GSSB and ULS data are also saved in the case of UTM-F.

TLS

The UTM TLS memory areas are created locally to the nodes in UTM cluster applications, as each TLS is assigned to an LTERM or (OSI-)LPAP and a connection can be established to every LTERM or (OSI-)LPAP in every cluster node at any time. A separate version of the memory area therefore exists in each cluster node.

5.4 Using users with RESTART=YES

In UTM cluster applications, a service restart is possible in all node applications for all genuine user IDs that have been generated with USER ..., RESTART=YES. Any genuine user generated with USER ...,RESTART=YES is always signed on exclusively, i.e. the user can only be signed on once to the UTM cluster application at any given time. Furthermore, such users can have no more than one open dialog service throughout the entire cluster.

The open services of such users can be continued in any node application provided that the open service is not bound to a node application, see below. A bound service can only be continued in the node application to which it is bound.

> **i** Note on UTM-F
> When a node application is terminated service data is lost for all services bound to this node. See next section "Node-bound services".

Node-bound services

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, following abnormal termination, an open service is bound to a node application if the user was signed on at the node application at the time the application was terminated.

If a user who wants to sign on at another node application even though his service is bound to a node application then the sign-on attempt is rejected if

- the node application to which the service is bound is running,
- or the bound service has a transaction in the state PTC (prepare to commit),
- or the UTM cluster application was generated with CLUSTER ... ABORT-BOUND-SERVICE = NO.

Connection user IDs

The service restart for connection user IDs of TS applications is bound to the connection and therefore to the node application. If a connection user ID has been generated by a TS application with RESTART=YES then it can have a service context that permits restarts in every node application.

5.5 Special issues

You must create the appropriate directories for storing the global cluster files before the generation run. These must exist and be accessible before the generation run.

The name for MAX KDCFILE and CLUSTER-NODE FILEBASE must not exceed 27 characters for UTM cluster applications.

5.6 Special issues with LU6.1 connections

More sessions (LSES statements) than connections (CON statements) can be assigned to an LPAP partner for a UTM cluster application. KDCDEF warns you of this with message K438, but a KDCFILE is created.

A node application is assigned to each session on generation (NODE-NAME operand in the LSES and CLUSTER-NODE statements). As a result, UTM can choose the "right" session when establishing a session with a partner application.

For information on what you need to consider for LU6.1 communication between a standalone application and a UTM cluster application, refer to the section "LU6.1-LPAP bundles of a standalone application with a UTM cluster application".

6 The KDCDEF generation tool

In order to generate a new UTM application or adapt an existing UTM application, you must first define the application configuration using KDCDEF control statements, and then use the KDCDEF generation tool to generate the UTM components KDCFILE and KDCROOT from which the UTM application program is created. For further information, see "Introduction to the generation procedure".

You can also modify the configuration of an application dynamically during operation. To do this, the RESERVE statement can be used during generation to reserve certain table locations for UTM objects. You can thus insert or remove clients, printers, user IDs, and services, KSETs, LTACs, CONs and LSESes in the configuration “on-the-fly” without affecting availability. The dynamic entry of objects is described in detail in the openUTM manual “Administering Applications”.

By issuing a CREATE-CONTROL-STATEMENTS statement during the KDCDEF run, you can read out the configuration information defined in the KDCFILE of a dynamically configured application, and convert this information to control statements. This function is known as inverse KDCDEF. The control statements which are generated in this way are written to a file which you can re-use directly as the input file for the KDCDEF run. For more information, see "Inverse KDCDEF".

6.1 Creating the ROOT table source, the KDCFILE and UTM cluster files

Based on the configuration information in the KDCDEF control statements, the KDCDEF generation tool creates the KDCFILE. This file contains all configuration and administrative data, as well as the ROOT table source for the main routine KDCROOT and - optionally - the UTM cluster files.

The KDCFILE, the ROOT table source and the UTM cluster files can be generated simultaneously in a single KDCDEF run or individually in separate KDCDEF runs. This is defined in the KDCDEF statement OPTION ..., GEN=.

All KDCDEF statements provided for defining the UTM application are listed in the following sections in accordance with their function group.

6.1.1 Statements for controlling the KDCDEF run

Statement	Function
EJECT	Initiate a page feed in the log
END	Terminate KDCDEF input
OPTION	Manage the KDCDEF run
REMARK or *	Insert a comment line
additional statement on BS2000 systems	
DEFAULT	Define default values

6.1.2 Statements for creating the ROOT table source

Statement	Function
AREA	Define names for additional data areas
EXIT	Define event exits
MAX	Define UTM application parameters
MESSAGE	Define the UTM message module
PROGRAM	Define program units
RESERVE	Reserve table locations for objects that can be entered dynamically
ROOT	Define a name for the ROOT table source
additional statement on BS2000 systems	
DATABASE	Define the database system (BS2000 systems)
FORMSYS	Define the format handling system
LOAD-MODULE	Define load modules for BLS
MPOOL	Define a common memory pool
TCBENTRY	Define a group of TCB entries
additional statements on Unix, Linux and Windows systems	
RMXA	Define a name for a resource manager on Unix, Linux and Windows systems (database connection via the X/Open XA interface)
SHARED-OBJECT	Define shared objects/DLLs for exchanging programs

6.1.3 Basic statements for creating a KDCFILE

Statement	Function
ACCOUNT	Define UTM accounting parameters
BCAMAPPL	Define additional application names for parallel connections
CREATE-CONTROL-STATEMENTS	Create control statements from the existing KDCFILE for a new KDCDEF run
KSET	Define a key set
LTERM	Define an LTERM partner as the logical access point for clients and printers
MAX	Define the name and runtime parameters of the UTM application
MESSAGE	Define a UTM message module
MSG-DEST	Define a user message line
PROGRAM	Define the names and properties of program units
PTERM	Define clients and printers
QUEUE	Reserve table entries for temporary message queues
RESERVE	Reserve table locations for objects that can be entered dynamically
SFUNC	Define special functions for the F and K keys
SIGNON	Control the sign-on procedure
SUBNET	Define IP subnets
TAC	Define the names and properties of transaction codes
TACCLASS	Define the number of processes for a TAC class
TAC-PRIORITIES	Define priorities for the TAC classes
TLS	Define a name for a TLS block
TPOOL	Define an LTERM pool
ULS	Define the names of ULS blocks
USER	Define user IDs

Statement	Function
additional basic statements on BS2000 systems	
DATABASE	Define the database system
EDIT	Define edit options
LOAD-MODULE	Define load modules for BLS
MPOOL	Define a common memory pool
MUX	Define a multiplex connection
SATSEL	Define SAT logging
additional basic statement on Unix, Linux and Windows systems	
CLUSTER	Define global properties of a UTM cluster application
CLUSTER-NODE	Define a node application of a UTM cluster
RMXA	Define a name for a resource manager
SHARED-OBJECT	Define shared objects/DLLs for exchanging programs

6.1.3.1 Creating the KDCFILE - additional statements for distributed processing via LU6.1

Statement	Function
BCAMAPPL	Define additional application names for parallel connections
CON	Define a logical connection to a UTM partner application
LPAP	Define an LPAP partner as the logical access point for a UTM partner application
LSES	Define a session name for the connection between two UTM applications
LTAC	Define local names for TACs in UTM partner applications
MASTER-LU61-LPAP	Define the master LPAP of an LU6.1-LPAP bundle
RESERVE	Reserve table locations for objects that can be entered dynamically (CON, LSES, LTAC)
SESCHA	Define the session characteristics
UTMD	Define the global values

6.1.3.2 Creating the KDCFILE - additional statements for distributed processing via OSI TP

Statement	Function
ABSTRACT-SYNTAX	Define the abstract syntax
ACCESS-POINT	Create an OSI TP access point for the local UTM application
APPLICATION-CONTEXT	Define the application context
LTAC	Define local names for TACs in UTM partner applications
MASTER-OSI-LPAP	Define a master LPAP for a OSI-LPAP bundle
OSI-CON	Define a logical connection to the partner application
OSI-LPAP	Define an OSI-LPAP partner as the logical access point for the partner application
RESERVE	Reserve table locations for objects that can be entered dynamically (LTAC)
TRANSFER-SYNTAX	Define the transfer syntax
UTMD	Define global values and the address of the local UTM application
additional statements on Unix, Linux and Windows systems	
MAX XAPTPSHMKEY	Define the key for the XAPTP shared memory segment
MAX OSISHMKEY	Define an authorization key for the OSS shared memory segment
MAX OSI-SCRATCH-AREA	Define the size of the working area for dynamic data storage

6.1.3.3 Generating KDCFILE and UTM cluster files - additional statements for UTM cluster applications

For UTM cluster applications on Unix, Linux, and Windows systems, you must also consider the following instructions:

Statement	Function
CLUSTER	Define global properties of a UTM cluster application
CLUSTER-NODE	Define a node application of a UTM cluster application
MAX [1] APPLIMODE ,APPLINAME ,GSSBS ,KB ,LSSBS ,NB ,ULS	Define UTM application parameters
OPTION GEN=(CLUSTER,..	Generate the UTM cluster files

[1]If the values of the operands listed here are modified then the UTM cluster files must be regenerated with OPTION GEN=(CLUSTER,...).

6.1.4 Effects of the KDCDEF statements on the generation objects

Not all statements of the KDCDEF generation tool have the same effect on the KDCFILE and ROOT table source. The table below shows which control statements affect which generation objects during the KDCDEF run:

KDCDEF control statement	KDCFILE	ROOT tables	KDCDEF control	Distributed processing via	
				LU6.1	OSI TP
ABSTRACT-SYNTAX	X				X
ACCESS-POINT	X				X
ACCOUNT	X				
APPLICATION-CONTEXT	X				X
AREA	X	X			
BCAMAPPL	X			X	
CON	X			X	
CREATE-CONTROL-STATEMENTS[1]					
EJECT			X		
END			X		
EXIT	X	X			
KSET	X				
LPAP	X			X	
LSES	X			X	
LTAC	X			X	X
LTERM	X				
MASTER-LU61-LPAP	X			X	
MASTER-OSI-LPAP	X				X
MAX[2]	X	X			X
MESSAGE	X	X			
MSG-DEST	X				

KDCDEF control statement	KDCFILE	ROOT tables	KDCDEF control	Distributed processing via	
				LU6.1	OSI TP
OPTION[3]	X	X	X	(X)	(X)
OSI-CON	X				X
OSI-LPAP	X				X
PROGRAM	X	X[4]			
PTERM	X				
QUEUE	X				
REMARK			X		
RESERVE	X	X[5]			
ROOT		X			
SESCHA	X			X	
SFUNC	X				
SIGNON	X				
SUBNET	X				
TAC	X				
TACCLASS	X				
TAC-PRIORITIES	X				
TLS	X				
TPOOL	X				
TRANSFER-SYNTAX	X				X
ULS	X				
USER	X				
UTMD	X			X	X

KDCDEF control statement	KDCFILE	ROOT tables	KDCDEF control	Distributed processing via	
				LU6.1	OSI TP
BS2000 specific statements					
DATABASE	X	X			
DEFAULT[6]	X	X	X		
EDIT	X				
FORMSYS	X	X			
LOAD-MODULE	X	X[7]			
MPOOL	X	X[8]			
MUX	X				
SATSEL	X				
TCBENTRY		X			
Unix, Linux and Windows system specific statements					
CLUSTER	X				
CLUSTER-NODE	X				
RMXA	X	X			
SHARED-OBJECT	X	X			

[1]Based on the configuration information defined in an existing KDCFILE, the CREATE-CONTROL-STATEMENTS statement generates an input file containing KDCDEF control statements for a new KDCDEF run.

[2]The operands CLRCH=, KB=, NB= and SPAB= only affect the generation of the ROOT table source. The other operands only affects the generation of the KDCFILE.

[3]The effect of the OPTION statement on the KDCFILE and the ROOT table source depends on the values entered for OPTION ...,GEN=.

[4]Only when generating a UTM application without load modules (on BS2000 systems), shared objects (on Unix and Linux systems) or DLLs (on Windows systems).

[5]Only when generating without the operand PROGRAM= and without load modules, shared objects or DLLs.

[6]The effect of the DEFAULT statement on the KDCFILE and the ROOT table source depends on the specified substatement.

[7]Only when extending the generation by *n* load modules.

[8]Only when generating without load modules.

The KDCDEF control statement OPTION...GEN= is used to define which objects (the KDCFILE, ROOT table sources and UTM cluster files) are to be generated by the KDCDEF generation tool.

When a new ROOT table source is created, this must be compiled (assembled on BS2000 systems) and relinked to your application. Relinking of an application program is only necessary if the table module is not dynamically loaded.
This is not necessary if you merely modify the KDCFILE. You can run the application with the new KDCFILE and the old main routine KDCROOT if no generation parameters that also affect KDCROOT have been changed when creating the new KDCFILE.

The MAX, ULS, CLUSTER and CLUSTER-NODE statements also affect the UTM cluster files during generation of UTM cluster applications. If you change parameters of the ULS, CLUSTER and/or CLUSTER-NODE statement when performing a new generation of a UTM cluster application, you must specify OPTION GEN=CLUSTER in order for the changes to take effect, see also "OPTION - manage the KDCDEF run".

6.2 Calling KDCDEF and entering the control statements

6.2.1 Starting KDCDEF and executing a KDCDEF run

> You can also start the KDCDEF run from WinAdmin. For further information, please see the WinAdmin online Help system, keyword „run KDCDEF“.

6.2.1.1 BS2000 systems

The KDCDEF generation tool is started using the command:

```
/START-KDCDEF
```

Alternatively, you can also call KDCDEF via the SDF command START-KDCDEF. This command is located in the SDF UTM application area. For more detailed information, see openUTM manual “Using UTM Applications on BS2000 Systems” section "Calling UTM tools".

KDCDEF reads the generation statements from SYSDTA from a SAM or ISAM file or from an LMS library element.. The control options for the KDCDEF run (see the OPTION statement, "OPTION - manage the KDCDEF run") are only processed by KDCDEF if it is read from SYSDTA. All other control statements for KDCDEF can be read from SYSDTA as well as from SAM or ISAM files or from an LMS library element.

The following restrictions apply to the use of LMS library elements:
Delta elements are not supported.

- The record type of read records is not evaluated.
- The records in the LMS elements may be a maximum of 256 characters in length.

The SAM or ISAM files or LMS library elements can be defined as input sources as described below:

- Assign an input file using the BS2000 commando ASSIGN-SYSDTA:

  ```
  /ASSIGN-SYSDTA TO-FILE=inputsource
  /START-KDCDEF
  ```

- Assign input files using the KDCDEF control statement OPTION ...,DATA=:

  ```
  /ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /START-KDCDEF
  OPTION DATA=inputsource1
  OPTION DATA=inputsource2
  etc.
  END
  ```

> You can catalog the files of the KDCFILE with the required attributes before calling the KDCDEF utility program. In particular, you can assign the volume and suitable primary and secondary allocation values as appropriate. If a KDCFILE file has already been cataloged then KDCDEF takes over the predefined attributes.
> If a file has not been cataloged then KDCDEF assigns the value 192 for both primary and secondary allocation when creating the file.

6.2.1.2 Unix and Linux systems

Proceed as follows to start the KDCDEF too and to execute a KDCDEF generation:

1. Add the directory below to the PATH environment variable:
   utm-path`/ex`.
   The *kdcdef* program used to start the KDCDEF generation tool is located in this directory.
2. Create one or more source files with an ASCII editor with control statements for the UTM generation. You must observe the information stated in section "Order of the control statements" and section "Format of the control statements".
3. Create the *filebase* directory (base directory of the application) in which openUTM stores the KDCFILE and other application-specific files. Enter the following command to create this directory:

   `mkdir` *filebase*

   You must create the directory **before** starting KDCDEF. *filebase* is the directory that you specified in the MAX statement in the FILEBASE= operand.
4. You start the KDCDEF tool with the *kdcdef* program.

   By default, KDCDEF reads the KDCDEF control statements from *stdin*. Only the control options for the KDCDEF run are read in from a shell script (see OPTION statement in section "OPTION - manage the KDCDEF run"), while the actual generation statements for KDCDEF are read from the files created in Step 2. You can specify these files directly at the start of KDCDEF:

   ```
   kdcdef < definput
   ```

   or after KDCDEF has been started using the KDCDEF statement OPTION:

   ```
   OPTION DATA=definput
   END
   ```

The messages and logs from KDCDEF are written to *stdout* and *stderr*, i.e. everything is displayed on the screen if you have not redirected the output.

You can redirect the output to a file as follows (you can select any name you want for the files):

`kdcdef < definput 2>`*def.err* `1>`*def.prot*

All UTM messages are recorded in *def.err* and *def.prot* contains the complete log of the KDCDEF run.

6.2.1.3 Windows systems

Proceed as follows to start the KDCDEF too and to execute a KDCDEF generation:

1. Add the directory below to the PATH environment variable: *utmpath*\ex.The *kdcdef.exe* program used to start the KDCDEF generation tool and other utility programs and DLLs are located in this directory. Proceed as follows:
   - In the Control Panel, open the dialog box *Environment Variables*. Possible access: In the search box enter the term *Environment Variables*, continue with *Edit the system environment variables* / *System Properties* / *Advanced* / *Environment Variables*.
   - Enter the path listed above to the PATH variable and click on the "Set" button.
2. Create one or more source files with control statements using an ASCII editor such as the NOTEPAD for the UTM generation. You must observe the information stated in section "Order of the control statements" and in section "Format of the control statements".
3. Create the *filebase* directory (project directory) in which openUTM stores the KDCFILE and other application-specific files. You must create the directory **before** starting KDCDEF. *filebase* is the directory that you specified in the MAX statement in the FILEBASE= operand.
4. Now start the KDCDEF tool. Open a command prompt window. KDCDEF reads the KDCDEF control statements from *stdin* by default, i.e. directly from the command prompt. Enter the following to have KDCDEF read the control statements from a file (e.g. *definput.txt*):

   ```
   kdcdef < definput.txt
   ```

   or start KDCDEF with `kdcdef` and pass the file using the KDCDEF statement OPTION:

   ```
   OPTION DATA=definput.txt
   ```

The messages and logs from KDCDEF are written to *stdout* and *stderr*, i.e. everything is displayed on the screen if you have not redirected the output. You can redirect the output to a file as follows (you can select any name you want for the files):

`kdcdef < definput.txt 2>`*def.err* `1>`*def.prot*

All UTM messages are recorded in *def.err* and *def.prot* contains the complete log of the KDCDEF run.

6.2.2 Order of the control statements

Apart from the following exceptions, the control statements can be entered in any order. Apart from END and UTMD, all control statements can be entered several times.

- The END statement is always specified last, and concludes the sequence of control statements.
- In the OPTION statement, the last parameter value specified always applies.
- The order of the AREA statements indicates the order in which these areas must be specified in the parameter list and declared in the program unit (e.g. in the LINKAGE-SECTION in COBOL). See the openUTM manual „Programming Applications with KDCS”.
- The sequence of the EXIT statements with USAGE=START and USAGE=SHUT defines the sequence in which the programs of the event exits START and SHUT are executed when the application is started or shut down.
- The master LTERM of a LTERM bundle must be generated before the slave LTERMs of this LTERM bundle.
- The primary LTERM of a LTERM group must be generated before the alias LTERMs of this LTERM group.
- The defined subnets are checked against the IP address of a connection request externally in the same order as that in which the SUBNET statements are entered.

BS2000 systems:

- The DEFAULT statement refers only to the control statements entered thereafter.
- Load modules are loaded in the same order as that in which the LOAD-MODULE statements are entered. Refer to the LOAD-MODULE statement in section "LOAD-MODULE - define a load module (BLS, BS2000 systems)" and openUTM manual “Using UTM Applications on BS2000 Systems”.

Unix, Linux and Windows systems:

- Shared objects/DLLs are loaded in the same order as that in which the SHARED-OBJECT statements are entered.

6.2.3 Format of the control statements

All KDCDEF control statements (apart from the DEFAULT statement on BS2000 systems) have the following format:

control-statement operand1, operand2,...

- *control-statement* can be entered starting in column 1 or later.
- *control-statement* must be separated from the operands by at least one blank.
- Each line of the control statement can be up to 240 characters in length. The control statements can be up to 3096 characters in length when continuation lines are used (see "Continuation lines in control statements").
- Comments can be inserted using the statement REMARK or by entering an asterisk (*) in column 1.
- The EJECT statement initiates a page feed in the log. The EJECT line itself is not logged.

6.2.4 Continuation lines in control statements

A control statement for the KDCDEF generation tool can consist of one or more lines, in which the hyphen (-) or backslash (\) can be used as the continuation character. In other words, if the last character of a line (apart from blanks) is a hyphen or a backslash, KDCDEF interprets the following line as belonging to the last statement specified. The continuation line can be entered starting in column 1 or later.

Each control statement can be up to 3096 characters in length, excluding comment lines, continuation characters, and blanks after the continuation character.

All comment lines must be marked with REMARK or an * in column 1.

6.2.5 Syntax and plausibility checks

KDCDEF carries out syntax and plausibility checks for all control statements entered. If KDCDEF does not detect any serious errors, then KDCDEF creates the KDCFILE and/or the source code for the ROOT tables, depending on what you have specified in OPTION.

In the case of UTM cluster applications, the UTM cluster files are also created where necessary.

KDCDEF always executes the plausibility checks for all control statements. If only one ROOT table source is created in a KDCDEF run, for example, then KDCDEF also checks the control statements that only affect the KDCFILE.

For this reason you should execute every KDCDEF run using all generation information, regardless of whether on the source code for the ROOT tables or only the KDCFILE is to be created.

Inconsistencies arising during the creation of the ROOT table module and KDCFILE that would otherwise only be detected once the application is started can be detected much earlier when complete plausibility checks are used. Errors are avoided.

6.2.6 KDCDEF logging

To improve legibility, KDCDEF logging can be structured as follows:

- Comments can be inserted in the KDCDEF log:
  - As a string surrounded by quotes:
    KDCDEF control statement "*comment*“
    The comment entered after a KDCDEF control statement must not contain quotes.
  - With * *comment* or **REM**ARK *comment*
    A * in column 1 or a REMARK statement create a comment line with a line number.
- Markers can be inserted in front of KDCDEF control statements, and must be preceded by a period (.*marker*). *marker* can be up to eight alphanumeric characters in length, and must begin with a letter.
- The EJECT statement initiates a page feed in the log. The EJECT line itself is not logged.

6.2.7 Format and uniqueness of object names

When configuring objects of the application, you must assign names to the objects. These names are then used by openUTM or the user to address specific objects. The following conditions should be borne in mind when assigning names:

- You must not use a reserved name.
- The object name must be unique within that particular object class.
- The name must not exceed the defined maximum length, and must contain permitted characters only.

6.2.7.1 Reserved names

Please note the comments below in order to ensure that the allocation of names does not result in unexpected, undefined UTM application behavior:

- Names which start with KDC are reserved for the transaction codes of the event services, the administration commands (KDCADM), the Dead Letter Queue and the SAT administration (BS2000 systems) and may only be used for such objects.This does not apply to the load modules belonging to a UTM application on BS2000 systems.
- On BS2000 systems program unit names must not start with prefixes which are used for runtime systems such as IT, IC etc.
- On Unix, Linux and Windows systems the names of UTM objects must not start with KDC, KC, x, ITS or mF. External names (e.g. program unit names) must not start with ’t_’, ’a_’, ’o_’ or ’s_’ which are reserved for CMX (t_) or OSS (a_, o_, s_).

6.2.7.2 Format of names

The following conventions must be observed for names entered in KDCDEF control statements:

- The base name of the KDCFILE (MAX ...,KDCFILE=) must comply with the rules for file names of the operating system, under which the application is to run (for further information, see MAX statement in section "MAX - define UTM application parameters").
- The names of LTERM partners, clients and printers, transaction codes and TAC queues etc. can be up to eight characters in length, where the following characters are permitted:
  - A,B,C,...,Z
  - 0,1,...,9
  - #, @, $

  On Unix, Linux and Windows systems, names may also contain lowercase letters (a,b,c,...,z). The names are case sensitive.
- Program names specified as entry/object names in the PROGRAM statement may be up to 32 characters long. This also applies for program names in TAC PROGRAM= and EXIT PROGRAM=.

  The following characters are permitted in program names:
  - A,B,C,...,Z
  - 0,1,...,9
  - #, @, $

  If other special characters are used, the program name must be enclosed in quotes, see below.
- Additional special characters in names:
  - Program names or passwords may also include other special characters such as "_" (underscore) and "-" (hyphen) if permitted by the particular system environment.
  - Names of IP subnets must start with an asterisk ("*").
  - Load module names on BS2000 systems (LOAD-MODULE) may also include the "." (period) and "-" (hyphen) characters.
  - Names that include special characters (program names, passwords, etc.) must be enclosed in quotes.
- Exceptions to name length rules:
  - Presentation and session selectors in the ACCESS-POINT and OSI-CON statements can be up to 16 characters in length.
  - Program names can be up to 32 characters long.
  - Processor names can be up to 64 characters long.
  - On BS2000 systems, load module names in BLS generation and character set names can be up to 32 characters long; the names of common memory pools (MPOOLs) can be up to 50 characters long.

6.2.7.3 Number of names

One name is created for each of the following control statements:

ACCESS-POINT
BCAMAPPL
CON
EDIT
KSET
LOAD-MODULE (BS2000 systems)
LPAP
LSES
LTAC
LTERM
MASTER-OSI_LPAP
MASTER-LU61-LPAP
MUX (BS2000 systems)
OSI-CON
OSI-LPAP
PROGRAM
PTERM
SHARED-OBJECT (Unix, Linux and Windows systems)
TAC
TLS
TPOOL
USER
ULS

Additional names are generated for the LTERM, MUX and TPOOL statements:

- If an application is generated without USER, two names are created for each LTERM statement.
- Two names are created for an LTERM statement belonging to a PTERM statement with PTYPE=APPLI, SOCKET, UPIC-R or UPIC-L if the implicit (connection) user belonging to this LTERM is not generated with an explicit USER statement.
- For each TPOOL statement, the number of names created is double that specified in the NUMBER= operand of the TPOOL statement. In the case of a TPOOL statement with PTYPE=APPLI, SOCKET, UPIC-R and UPIC-L the number of names created is **threefold** that specified in NUMBER=.
- Two names are created for each MUX statement.

Unix, Linux and Windows systems:

Additional names are generated for the CLUSTER and CLUSTER-NODE statements:

- One BCAMAPPL is also generated for the CLUSTER statement.
- One PTERM, one LTERM and one USER are also generated for each CLUSTER-NODE statement.

Furthermore, up to six additional names are created during generation, which are required by openUTM for event services (KDCSGNTC, KDCBADTC, KDCMSGTC, KDCMSGUS, KDCMSGLT, KDCAPLKS). The first three names can also be specified in a TAC statement. The last three names may not be specified.

If XATMI program units are generated for a UTM application, i.e. if API=(XOPEN,XATMI) is set in at least one TAC statement, then a TAC entry named KDCTXCOM and a PROGRAM entry named KDCTXRLB are created by openUTM.

The name KDCDLETQ is created for the dead letter queue during generation. The properties of this TAC queue can also be defined in a separate TAC statement.

Maximum values for names

The table below shows the maximum number of names that can be created using KDCDEF control statements. If this number is exceeded, then the generation is terminated.

Group of KDCDEF control statements	Maximum number of generated names
BS2000 systems	
#PTERM + #CON + TPOOLNR + #OSI-ASSOCIATIONS + #MUX[1]	<= 500 000
#LTERM + #LPAP + TPOOLNR + #OSI-LPAP + #TASKS + #MUX[1] + 1	<= 500 000
Unix, Linux and Windows systems	
#PTERM + #CON + TPOOLNR + #OSI-ASSOCIATIONS	<= 500 000
#LTERM + #LPAP + TPOOLNR + #OSI-LPAP + #TASKS + 1	<= 500 000
BS2000, Unix, Linux and Windows systems	
#USER + #APPLI + #LSES + #OSI-ACTIVE-ASSOCIATIONS + (2 * #TASKS) + 1	<= 500 000
#PROGRAM	<= 32 000
#TAC + 4	<= 32 000
#LSES	<= 65 000
#CON	<= 65 000
#KSET + 1	<= 32 000
#LTAC	<= 32 000
#MUX[1]	<= 9 999
Total of all other names + 2	<= 32 767

[1]Only on BS2000 systems

Description of placeholders:

#*statement* Number of names generated using this KDCDEF statement

#APPLI Number of PTERM statements plus the TPOOLNR values of the TPOOL statements with PTYPE=APPLI/SOCKET/UPIC-R and UPIC-L (UPIC-L only on Unix, Linux and Windows systems)

In the case of UTM cluster applications, the values of #PTRM, #LTRM and #APPLI are each increased by the number of specified CLUSTER-NODE statements.

#MUX Total number of generated MUX statements (only on BS2000 systems)

#OSI-ACTIVE-ASSOCIATIONS

Number of active parallel OSI connections of the generated operand values (OSI-CON ..., ACTIVE=YES and associated OSI-LPAP...,ASSOCIATIONS=*number*). This is the sum of all ASSOCIATIONS values in all OSI-LPAP statements.

#OSI-ASSOCIATIONS

#OSI-ACTIVE-ASSOCIATIONS plus the number of inactive parallel OSI-connections. (OSI-CON ..., ACTIVE=YES/NO and associated OSI-LPAP...,ASSOCIATIONS=*number*). This is the sum of all ASSOCIATIONS values in all OSI-LPAP statements, including the values of OSI-LPAP statements for which backup connections are generated.

TPOOLNR Sum of all NUMBER= operands (number of LTERM partners in each LTERM pool) in all generated TPOOL statements

The following must also be noted:

- The number of names for #PROGRAM, #TAC, #LTERM, #PTERM, #USER, #KSET and #LTAC includes names generated statically and reserved names for objects that can be entered dynamically.
- The names of MASTER-LU61-LPAP statements must also be counted with #LPAP.
- The names of MASTER OSI-LPAP statements must also be counted for #OSI-LPAP.
- If the application was generated without USER statements, #USER must be replaced by #LTERM + TPOOLNR in the first condition.
- You can generate up to 100 ULS blocks and 100 TLS blocks.
- The number of generated user IDs (#USER) plus the number of entries intended for service stacking (defined in MAX NRCONV=) is restricted to a maximum of 500000.
- The number of generated user IDs (#USER) plus the number of entries intended for service stacking (MAX NRCONV) plus the maximum number of possible parallel asynchronous services (defined in MAX ASYNTASKS = (...,*service_number*)) plus the number of entries reserved for sign on services (SIGNON CONCURRENT-TERMINAL-SIGNON) is restricted to a maximum of 665000.

6.2.7.4 Uniqueness of names and addresses

The objects of a UTM application are combined in shared name spaces which are defined for specific object types. The names and address of objects of the permitted types must be unique throughout the name class. A name or address must only be assigned once within the name class. There are three name classes:

Name class 1

- LTERM partners (statement LTERM *ltermname*)
- LTERM partners created by openUTM for the LTERM pools (statement TPOOL ...,LTERM=*ltermprefix*, NUMBER=*number*)
- transaction codes and TAC queues (statements TAC *tacname*)
- LPAP or OSI-LPAP partners for server-to-server communication (statements OSI-LPAP *osi_lpap_name* and LPAP *lpapname*)

Name class 2

- user IDs (statement USER *username*)
- sessions for distributed processing based on LU6.1 (statement LSES *sessionname*)
- connections and associations for distributed processing based on OSI TP (statement OSI-LPAP..., ASSOCIATION-NAMES=, ASSOCIATIONS=)

Name class 3

- clients and printers (PTERM statement)
  Clients are terminals, UPIC-clients, transport system applications (DCAM, CMX and socket applications), and UTM partner applications that do not use a higher-level protocol (LU6.1, OSI TP) during communication.
- name of the partner application for distributed processing based on LU6.1 (CON statement)
- name of the partner application for distributed processing based on OSI TP (OSI-CON statement)
- multiplex connections of a UTM application on BS2000 systems (MUX statement)

The objects listed in name class 3 are communication partners of the UTM application. openUTM must be able to uniquely identify these objects and the connections to them. For this purpose, it assigns a name triplet to each communication partner. This name triplet must be unique within the UTM application and consists of the following components:

- the name of the communication partner.
  This is specified in *ptermname* in the PTERM statement, in *remote_appliname* in the CON statement, in TRANSPORT-SELECTOR= in the OSI-CON statement and in *name* in the MUX statement.
  On BS2000 systems the BCAM name of the communication partner must be specified.
- the name of the system on which the communication partner is located.This is specified in the PRONAM= operand of the PTERM, CON and MUX statements and in the NETWORK-SELECTOR= operand of the OSI-CON statement.
- the name of the local application via which the connection to the communication partner is established. This is specified in the BCAMAPPL= operand of the PTERM, MUX and CON statements and in the LOCAL-ACCESS-POINT= operand of the OSI-CON statement.

6.2.8 Result of the KDCDEF run

Depending on the entries made during generation, the KDCDEF generation tool creates the following:

- *BS2000 systems:*
  - the KDCFILE with the main file *filebase*.KDCA and, if dual-file operation is used, the duplicate file *filebase*.KDCB
  - the ROOT table source
  - the page pool *filebase*.P*nn*A, possibly with the duplicate *filebase*.P*nn*B
  - the restart area *filebase*.R*nn*A, possibly with the duplicate *filebase*.R*nn*B
- *Unix, Linux and Windows systems:*
  - the main file KDCA in the *filebase* directory and, if dual-file operation is used, the duplicate file KDCB, also in the *filebase* directory
  - the ROOT table source in the form of a C/C++ source
  - the page pool P*nn*A, possibly with the duplicate P*nn*B, in the *filebase* directory
  - the restart area R*nn*A, possibly with the duplicate R*nn*B, in the *filebase* directory
- Additionally, if a UTM cluster application is being generated, see also the section "Notes on generating a UTM cluster application on Unix, Linux and Windows systems":
  - the cluster configuration file
  - the cluster user file
  - the cluster page pool files (a control file and one or more files for the user data)
  - the cluster GSSB file
  - the cluster ULS file

The format of the KDCFILE is described in detail in section "The KDCFILE".

KDCDEF outputs a message to SYSOUT (BS2000 systems) or *stderr* (Unix, Linux and Windows systems) indicating whether the KDCFILE was created successfully and specifying the size of the KAA (KDC Application Area) occupied by the application. It also outputs a log containing the control statements and any error messages to SYSLST (BS2000 systems) or *stdout* (Unix, Linux and Windows systems).

Note for KDCDEF on BS2000 systems

If the KDCDEF generation tool terminates abnormally due to an error, it sets process switch 3 (as occurs with all UTM tools). In this case, no files are generated apart from those created by the CREATE-CONTROL-STATEMENTS statement.

6.3 Inverse KDCDEF

The inverse KDCDEF function provided by openUTM is used to ensure that changes made to the configuration dynamically during runtime are not lost when your application is regenerated. It creates control statements for the KDCDEF generation tool from the configuration data in the current KDCFILE.

Inverse KDCDEF generates control statements for object types that can be entered and deleted dynamically:

- **USER statements**

  For all user IDs currently defined in the application. Inverse KDCDEF does not create USER statements for user IDs defined internally by UTM for the LTERM partners of clients of type UPIC-R, APPLI and SOCKET.

- **LTERM statements**

  For all LTERM partners of the application which do not belong to an LTERM pool or to a multiplex connection (BS2000 systems).

- **PTERM statements**

  For all clients and printers entered in the configuration. No PTERM statements are created for clients that connect via an LTERM pool to the application or that belong to a multiplex connection.

- **PROGRAM statements**

  For all program units and conversation exits currently defined in the application configuration.

- **TAC statements**

  For all transaction codes and TAC queues of the application.

- **KSET statements**

  For all key sets of the application.

- **CON statements**

  For all LU6.1 connections of the application.

- **LSES statements**

  For all LU6.1 session names of the application.

- **LTAC statements**

  For all local transaction codes for VTV partner applications.

Control statements are also generated for objects of the types listed above, which were created statically in a previous KDCDEF generation. All modifications entered dynamically for these objects during runtime are taken into consideration.

Inverse KDCDEF does **not** create control statements for object types other than those listed above. Nor does it generate control statements for other components of the application or for application parameters.

It does **not** create control statements for objects that were dynamically deleted from the application configuration. After regeneration, these objects are thus permanently removed from the configuration. They do not occupy a table location and their object names are no longer reserved.

After regeneration with KDCDEF, the update tool KDCUPD does not transfer any application data from the old KDCFILE to the new KDCFILE, which relates to objects deleted dynamically. This applies even if the new KDCDEF generation includes an object with the same name and type as a deleted object. In particular, KDCUPD does not transfer any asynchronous jobs created by LTERM partners or user IDs that have since been deleted.

The USER statements created by inverse KDCDEF do not include any passwords. For user IDs generated with a password, inverse KDCDEF creates USER statements with the following format:

```
USER username, PASS=*RANDOM,....
```

Once the KDCDEF run is complete and the new KDCFILE has been created, you must transfer the passwords of the user IDs to the new KDCFILE using the KDCUPD tool. This is also possible in a UTM-F application. For further information, see "The tool KDCUPD – updating the KDCFILE".

It is not generally necessary to transfer the passwords with KDCUPD with UTM cluster applications. In UTM cluster applications, the current passwords are stored in the cluster user file and not in the KDCFILE.

You only need to transfer the passwords with KDCUPD if a new cluster user file has been generated and you wish to retain the passwords from the last application run.

> **i** In order to ensure that the KDCFILE contains the current passwords, the current information on all users must be read once (e.g. using WinAdmin or WebAdmin.) before the application is terminated.

6.3.1 Starting inverse KDCDEF

Inverse KDCDEF can be started online or offline.

The inverse KDCDEF run is started **online** by issuing the KC_CREATE_STATEMENTS call via the program interface for administration. Further information can be found in the openUTM manual “Administering Applications”. Inverse KDCDEF can only be started **offline** if the application is not running, i.e. outside the application runtime. Since inverse KDCDEF reads data from the KDCFILE, you must ensure that this data is not modified during the inverse KDCDEF run.

Inverse KDCDEF can be started offline by calling the KDCDEF generation tool and issuing the control statement **CREATE-CONTROL-STATEMENTS**. This statement is described in section "CREATE-CONTROL-STATEMENTS - Create KDCDEF control statements".

You can start inverse KDCDEF such that KDCDEF control statements are created either for all permitted object types, or only for those object types combined in the object groups CON, DEVICE, KSET, LSES, LTAC, PROGRAM and USER.

- `CREATE-CONTROL-STATEMENTS *ALL`

  KDCDEF control statements are created for all objects of type TAC, PROGRAM, PTERM, LTERM USER, KSET, LTAC, CON and LSES.
- `CREATE-CONTROL-STATEMENTS DEVICE`

  LTERM and PTERM statements are created for LTERM partners, clients and printers.
- `CREATE-CONTROL-STATEMENTS PROGRAM`

  PROGRAM and TAC statements are created for program units, conversation exits, and transaction codes.
- `CREATE-CONTROL-STATEMENTS USER`

  USER statements are created for user IDs.
- `CREATE-CONTROL-STATEMENTS KSET`

  KSET statements are created for key sets.
- `CREATE-CONTROL-STATEMENTS LTAC`

  LTAC statements are created for transaction codes. These are used to start the service programs in partner applications.
- `CREATE-CONTROL-STATEMENTS CON`

  CON statements are created for transport connections to remote LU6.1 applications.
- `CREATE-CONTROL-STATEMENTS LSES`

  LSES statements are created for assigning new LU6.1 session names.

6.3.2 Result of inverse KDCDEF

With inverse KDCDEF, you can define that

- all control statements are to be written to a file or - on BS2000 systems - to an LMS library element.
- or that the control statements of a particular object group are to be written to a separate file or - on BS2000 systems - to a separate LMS library element.

When starting inverse KDCDEF, you specify the name(s) of the file(s) or LMS library element(s) to be created. If a file or LMS library element with this name does not exist, the file or library element is created automatically. If a file or LMS library element with this name already exists. It is created automatically. Otherwise you can define whether it is to be overwritten or updated.

The CREATE-CONTROL-STATEMENTS statement is applied immediately. You can therefore issue the OPTION statement immediately after the CREATE-CONTROL-STATEMENTS statement in the same KDCDEF run. This transfers the files created by inverse KDCDEF to KDCDEF. For example:

```
CREATE-CONTROL-STATEMENTS *ALL, TO-FILE=control_statements_file
                                ,MODE=CREATE,FROM-FILE=kdcfile
OPTION DATA=control_statements_file
```

The diagram below illustrates how you can transfer the files generated by inverse KDCDEF directly as input files to KDCDEF. However, you can also edit them, i.e. modify them before the KDCDEF run and pass them to KDCDEF later as part of a regeneration. In this case, you simply terminate the generated control statements with the END statement. You assign each generated input file to KDCDEF with the control statement OPTION DATA= *control_statements_file* before the start.

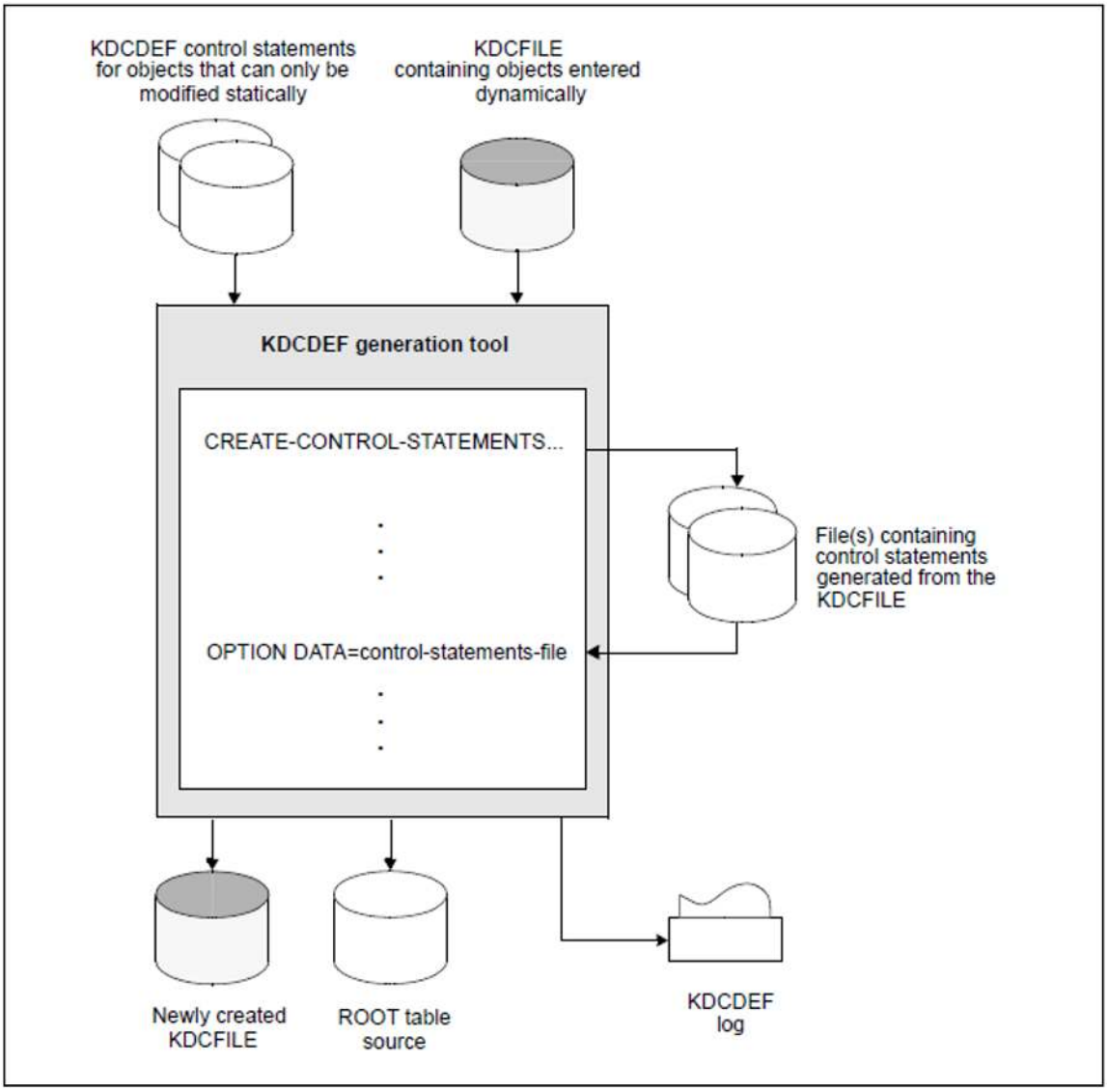


Figure 18: KDCDEF run with inverse KDCDEF

6.3.3 Creating KDCDEF control statements in upgrades

To enable inverse KDCDEF to read information from the KDCFILE, you must ensure that the KDCFILE was created with the same openUTM version as the KDCDEF generation tool used for the inverse KDCDEF run.

If you upgrade to a new openUTM version, the KDCDEF control statements must first be created in the previous version, i.e. you must start the inverse KDCDEF of the previous version. The generated files can then be used as input files for the KDCDEF of the new openUTM version.

6.4 Recommendations when regenerating an application

During the operation of a UTM application, it may become necessary to regenerate the application.

For UTM cluster applications on Unix, Linux or Windows systems, there are changes that can be made with a new generation of the KDCFILE with a running UTM cluster application and changes that can only be made when the UTM cluster application has been completely terminated.

A list of changes that require the UTM cluster application to be completely terminated before the application is started with the new KDCFILE can be found in the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”

Possible reasons for initiating a new KDCDEF run are listed below:

- to adjust the maximum values defined during generation
- to create new objects for distributed processing based on LU6.1 or OSI TP, because the server group is to be expanded during distributed processing A KDCDEF run is only needed for distributed processing based on LU6. 1 if it is necessary to insert LPAP objects. Objects of the types CON, LSES and LTAC, on the other hand, can be created with dynamic administration (provided that sufficient table entries were reserved with the RESERVE statement).
- to enter new load modules (BS2000 systems), shared objects (Unix and Linux systems) or DLLs (Windows systems) in the application program
- in cases where table locations reserved for the dynamic entry of objects in the configuration are occupied, to extend the table or to remove objects marked for deletion in order to release the table locations and object names for further use

The application downtime associated with regeneration can be reduced by observing the following recommendations:

- When generating your application for the first time, split the KDCDEF control statements between various files depending on whether the objects involved can only be generated statically or can be entered dynamically. These files can then be provided to KDCDEF as input files using the OPTION DATA= statement.
- The control statements USER, LTERM, PTERM, PROGRAM TAC, CON, KSET, LSES and LTAC should be entered separately in files in accordance with the various object groups. When regenerating the application, you can simply replace these files with those created by inverse KDCDEF (DEVICE, PROGRAM, and USER, CON, KSET, LSES and LTAC). Further information can be found in section "CREATE-CONTROL-STATEMENTS - Create KDCDEF control statements".

Before regenerating the application or initiating the inverse KDCDEF run, it is recommended that you dynamically delete all objects that are to be excluded from the new configuration (KC_DELETE_OBJECT call). Further information can be found in the openUTM manual “Administering Applications”.

> **i** In UTM cluster applications, objects that can be administered dynamically must always be deleted using the administration facilities. Only deleting the objects in the KDCDEF source leads to inconsistencies in the individual node applications of the UTM cluster application.

Compared to the manual deletion of control statements from the input file for the KDCDEF run, dynamic deletion offers the following advantages:

- If an object is manually deleted from the input file during regeneration, and another object is defined with the same name and type but with different properties in the same generation run, the KDCUPD tool does not recognize these as two different objects, and transfers the data of the deleted object to the KDCFILE. This can be avoided by dynamically deleting the object beforehand, and then creating an object with the same name and type during regeneration. In this case, KDCUPD will recognize these as two different objects, and will not transfer the data of the old object into the new KDCFILE.
- The manual deletion of KDCDEF statements from the KDCDEF input file is both tedious and prone to errors. During deletion, you must look out for dependencies between objects and thus between the KDCDEF statements. If dependencies are inadvertently overlooked, the KDCDEF run will have to be repeated thus increasing downtimes.
- The processes performed during regeneration can be automated. Within a single procedure, you can call inverse KDCDEF, transfer the generated files directly to KDCDEF, and call the KDCUPD update tool. This fully automatic procedure minimizes downtimes during regeneration.

To prevent undesirable repercussions from dynamic deletion, make sure for instance that there are no jobs pending for objects deleted or loaded dynamically during runtime.

6.5 KDCDEF control statements

- ABSTRACT-SYNTAX - define the abstract syntax
- ACCESS-POINT - create an OSI TP access point
- ACCOUNT - define the accounting functions
- APPLICATION-CONTEXT - define the application context
- AREA - define additional data areas
- BCAMAPPL - define additional application names
- CHAR-SET- assign names to code tables (BS2000 systems)
- CLUSTER - define global properties of a UTM cluster application (Unix, Linux and Windows systems)
- CLUSTER-NODE - define a node application of a UTM cluster application (Unix, Linux and Window systems)
- CON - define a connection for distributed processing based on LU6.1
- CREATE-CONTROL-STATEMENTS - create KDCDEF control statements
- DATABASE - define a database system (BS2000 systems)
- DEFAULT - define default values (BS2000 systems)
- EDIT - define edit options (BS2000 systems)
- EJECT - initiate a page feed in the log
- END - terminate KDCDEF input
- EXIT - define event exits
- FORMSYS - define the format handling system (BS2000 systems)
- HTTP-DESCRIPTOR - define a HTTP Descriptor
- KSET - define a key set
- LOAD-MODULE - define a load module (BLS, BS2000 systems)
- LPAP - define an LPAP partner for distributed processing based on LU6.1
- LSES - define a session name for distributed processing based on LU6.1
- LTAC - define a transaction code for the partner application
- LTERM - define an LTERM partner for a client or printer
- MASTER-LU61-LPAP - define the master LPAP of an LU6.1-LPAP bundle
- MASTER-OSI-LPAP - defining the master LPAP of an OSI-LPAP bundle
- MAX - define UTM application parameters
- MESSAGE - define a UTM message module
- MPOOL - define a common memory pool (BS2000 systems)
- MSG-DEST - define user-specific messages destinations
- MUX - define a multiplex connection (BS2000 systems)
- OPTION - manage the KDCDEF run
- OSI-CON - define a logical connection to an OSI TP partner
- OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP
- PROGRAM - define a program unit
- PTERM - define the properties of a client/printer and assign an LTERM partner

- QUEUE - reserve table entries for temporary messages queues
- REMARK - insert a comment line
- RESERVE - reserve table locations for UTM objects
- RMXA - define a name for an XA database connection (Unix, Linux and Windows systems)
- ROOT - define a name for the ROOT table source
- SATSEL - define SAT logging (BS2000 systems)
- SESCHA - define session characteristics for distributed processing based on LU6.1
- SFUNC - define function keys
- SHARED-OBJECT - define shared objects/DLLs (Unix, Linux and Windows systems)
- SIGNON - control the sign-on procedure
- SUBNET - define IP subnets
- TAC - define the properties of transaction codes and TAC queues
- TACCLASS - define the number of processes for a TAC class
- TAC-PRIORITIES - specify priorities of the TAC classes
- TCBENTRY - define a group of TCB entries (BS2000 systems)
- TLS - define a name for a TLS block
- TPOOL - define an LTERM pool
- TRANSFER-SYNTAX - define the transfer syntax
- ULS - define a name for a ULS block
- USER - define a user ID
- UTMD - application parameters for distributed processing

6.5.1 ABSTRACT-SYNTAX - define the abstract syntax

The ABSTRACT-SYNTAX control statement is only required if you want to define your own Application Context for communication via the OSI-TP protocol (see the APPLICATION-CONTEXT statement in section "APPLICATION-CONTEXT - define the application context").

ABSTRACT-SYNTAX defines a local name for an abstract syntax, and to assign an object identifier and the transfer syntax selected for transferring the user data. Since openUTM automatically generates the abstract syntaxes CCR, UDT, XATMI and UTMSEC. Therefore, they need not be explicitly generated using the ABSTRCT-SYNTAX statement. It is possible to generate up to 50 abstract syntaxes, including those generated implicitly by openUTM.

ABSTRACT-SYNTAX	abstract_syntax_name **,OBJECT-ID**ENTIFIER=object_identifier [,TRANSFER-SYNTAX=transfer_syntax_name]

abstract_syntax_name

Local name for an abstract syntax up to eight characters in length. This name must be unique within the UTM application.

abstract_syntax_name must be specified in MGET/MPUT or FGET/FPUT when sending or receiving data in this abstract syntax.

OBJECT-IDENTIFIER=object_identifier

Object identifier of the abstract syntax specified as follows:

object_identifier=(number1,number2, ... ,number10)

number is a positive integer in the range 0 to 67108863. For *object_identifier*, you can specify two to ten integers enclosed in parentheses, each of which is separated by a comma. The number of integers entered and their positions are relevant.

Instead of the integer itself, you can also specify the symbolic name assigned to this integer. The table in section "OSI terms" shows the permitted values for *number* at the various positions.

object_identifier must be unique with the UTM application, i.e. another abstract syntax must not be generated with the same object identifier.

TRANSFER-SYNTAX=transfer_syntax_name

Name of a transfer syntax defined using the TRANSFER-SYNTAX control statement.

Default: BER (Basic Encoding Rules)

openUTM automatically generates the abstract syntaxes CCR, UDT, XATMI and UTMSEC, which are defined as follows:

Generation of “CCR”:

```
ABSTRACT-SYNTAX CCR,                                -
                OBJECT-IDENTIFIER=(2, 7, 2, 1, 2),  -
                TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

```
(joint-iso-ccitt, ccr, abstract-syntax, apdus, version2)
```

Generation of “UDT”:

```
ABSTRACT-SYNTAX UDT,                                    -
                OBJECT-IDENTIFIER=(1, 0, 10026, 6, 1, 1),  -
                TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

```
(iso, standard, tp, udt, generic-abstract-syntax, version)
```

Generation of “XATMI”:

```
ABSTRACT-SYNTAX XATMI,                                      -
                OBJECT-IDENTIFIER=(1, 2, 826, 0, 1050, 4, 1, 0),  -
                TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

```
(iso, national-member-body, bsi, disc, xopen, xatmi, apdus-abstract-syntax,
version1
```

Generation of “UTMSEC”:

```
ABSTRACT-SYNTAX UTMSEC,                                        -
                OBJECT-IDENTIFIER=(1, 3, 0012, 2, 1107, 1, 6, 1, 2, 0), -
                TRANSFER-SYNTAX=BER
```

Symbolic representation of the object identifier:

```
(iso, identified-organisation, icd-ecma, member-company, siemens-units, sni,
transaction-processing, utm-security, abstract-syntax, version)
```

6.5.2 ACCESS-POINT - create an OSI TP access point

The ACCESS-POINT control statement is required only for communication based on the OSI TP protocol. It defines a local access point to the services of OSI TP.

> If you issue more than one ACCESS-POINT statement per application, then KDCDEF outputs warning K492.
>
> If more than one ACCESS-POINT statement is generated for an application then the applcation program have to make sure that all partner applications involved in a single transaction are connected to the UTM appilcation via the same service access point. OSI TP does not support tansactions spreading over more than one service access point.

Using the information specified in the ACCESS-POINT statement, a partner application can address the local application.

You specify the following parameters for a service access point in the ACCESS-POINT statement:

- Address of the access point within the local system
  The address of the access points consists of the presentation selector, session selector and transport selector components.

  The address specifications must be coordinated with the communication partners. The TRANSPORT-SELECTOR specification is mandatory in all cases.

  Unix, Linux and Windows systems:
  On Unix, Linux and Windows systems the address of the access point also comes from the LISTENER-PORT, T-PROT, and TSEL-FORMAT components.
  See "Providing address information for the CMX transport system (Unix, Linux and Windows systems)" for more information.
  Only a maximum of 1000 connections can be established per access point at a time. If you require more concurrent connections in your application, you must define more than one access point. But in this case note the above info box.
- Application Entity Qualifier
  You can define an application entity qualifier (AEQ) as additional address information. The application entity qualifier (AEQ) is combined with the application process title (APT) defined in the UTMD statement to form the application entity title (AET). The AET is a globally unique name for an application entity within the OSI TP environment. During transaction-oriented processing, the partner application requires the AET of the local UTM application in order to establish a connection. Similarly, the local application requires the AET of the partner application. It must be specified in the OSI-LPAP control statement that defines the partner application.
  The transport selector for the access point is still a mandatory entry.
- Listener ID (Unix, Linux and Windows systems)
  On Unix, Linux and Windows systems the access point is assigned a listener ID.

Each ACCESS-POINT is signed on to the transport system when the application is started (provided this is possible), and is not signed off until the application is terminated.

ACCESS-POINT	access_point_name [,**A**PPLICATION-**E**NTITY-**Q**UALIFIER=aequalifier] ,**P**RESENTATION-**SEL**ECTOR={ *NONE | (C'c' [,STD | EBCDIC | ASCII]) | X'x' } ,**S**ESSION-**SEL**ECTOR={ *NONE | (C'c' [, STD | EBCDIC | ASCII]) | X'x' } ,**T**RANSPORT-**SEL**ECTOR=C'c' ***further operands for Unix, Linux and Windows systems*** [,LISTENER-ID=number] [,LISTENER-PORT=number] [,T-PROT=(RFC1006)] [,TSEL-FORMAT={ T | E | A }]

access_point_name

Name of the OSI TP access point, which is then used to identify the access point in the local UTM application.

access_point_name can be up to eight characters in length. *access_point_name* must be unique within the local UTM application.

APPLICATION-**E**NTITY-**Q**UALIFIER=aequalifier

Address component of the application entity title (AET). The AET is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET in order to establish a connection.

An application entity qualifier (AEQ) can be specified only if an application process title (APT) is also defined for the application in the UTMD statement.

However, an APT need not necessarily be assigned an AEQ. If AEQ is **not** defined , the access point has no application entity title (AET), i.e. it cannot be used for transaction management (commit functional unit).

> **i** If the application context of an OSI-LPAP partner that operates via this access point (OSI-CON statement) contains the CCR syntax, you must enter an application entity qualifier here.

For *aequalifier*, specify a positive integer. *aequalifier* must be unique within the application, i. e. *aequalifier*=integer1 must not be specified as the AEQ in any other ACCESS-POINT statement.

Minimum value: 1
Maximum value: 67 108 863 (2^{26}-1)

LISTENER-ID= number

This operand is supported only on Unix, Linux and Windows systems.

This assigns a listener ID to the access point as administrative information.

Listener IDs can be specified for access points and application names. See also the BCAMAPPL statement in section "BCAMAPPL - define additional application names".

You can use the listener IDs to distribute the network connections of the access points to different network processes. All connections of an access point are managed by the same network process.

If you do not explicitly specify a listener ID, openUTM assigns the value 0 and combines all connections without a listener ID into a single network process.

Default value: 0
Minimum value: 0
Maximum value: 65535

BCAMAPPL names that were created for communication via the socket interface (native TCP/IP) use separate network processes. Their listener IDs comprise a separate number space. i.e. they are administered in a different network process even if they have the same listener ID as this access point.

LISTENER-PORT= number

This operand is supported only on Unix, Linux and Windows systems.

Port number of the access point.

All port numbers between 1 and 65535 are allowed.

Default: 0 (i.e. no port number)

If OPTION CHECK-RFC1006=YES, then a port number must be entered for LISTENER-PORT.

PRESENTATION**-SEL**ECTOR=

Presentation selector for the address of the OSI TP access point.

*NONE The address of the OSI TP access point does not contain a presentation selector.

C’c’ The presentation selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The presentation selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted.

STD The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix, Linux and Windows systems = ASCII).

Default: STD

EBCDIC The characters are interpreted as EBCDIC code.

ASCII — The characters are interpreted as ASCII code.

X’x’ — The presentation selector is entered in the form of a hexadecimal number (x). The value specified for *x* can be up to 32 hexadecimal digits (corresponds to 16 bytes) in length. You must enter an even number of hexadecimal digits.

SESSION-**SEL**ECTOR=

Session selector for the address of the OSI TP access point.

*NONE — The address of the OSI TP access point does not contain a session selector.

C’c’ — The session selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The session selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted.

STD — The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix, Linux and Windows systems = ASCII).

Default: STD

EBCDIC — The characters are interpreted as EBCDIC code.

ASCII — The characters are interpreted as ASCII code.

X’x’ — The session selector is entered in the form of a hexadecimal number (x).
The value specified for *x* can be up to 32 hexadecimal digits (corresponds to 16 bytes) in length. You must enter an even number of hexadecimal digits.

TRANSPORT-**SEL**ECTOR=C’c’

Transport component for the address of the OSI TP access point.

The specification of T-SEL=C’c’ is mandatory.

You can enter up to eight printable characters. Permitted characters include uppercase letters, numbers, and the special characters $, # and @. Hyphens are not permitted. The first character of the name must be an uppercase letter.

The name defined in T-SEL must be unique in the local UTM application. It must not be the same name as the primary application name specified in MAX APPLINAME, a BCAMAPPL name or the name specified with a Tselector in an ACCESS-POINT control statement.

BS2000 systems:

T-SEL= specifies the local BCAM application name.The transport selector must be unique in the local system for each host.

Unix, Linux and Windows systems:

You must match T-SEL to the transport selector of the OSI TP partner. If, for example, the partner is a UTM application, the specification in T-SEL must match the transport selector of the OSI-CON statement on the partner.

T-PROT= Address formats of the T-selectors of the access point

This operand is supported only on Unix, Linux and Windows systems.

Further Information, see "PCMX documentation" (openUTM documentation).

RFC1006 Address format RFC1006, ISO transport protocol based on TCP/IP and RFC1006 convergence protocol.

Default: RFC1006

TSEL-FORMAT= Format indicator of the T-selectors of the access point (operand TRANSPORT-SELECTOR)

This operand is supported only on Unix, Linux and Windows systems.

The format indicator specifies the encoding of the T-selectors in the transport protocol. You will find more information in the “PCMX documentation” (openUTM documentation).

T TRANSDATA format (encoded in EBCDIC)

E EBCDIC character format

A ASCII character format

Default:

- T if the character set of the T-selector corresponds to the TRANSDATA format
- E in all other cases

It is recommended to specify a value explicitly for TSEL-FORMAT.

6.5.3 ACCOUNT - define the accounting functions

The ACCOUNT control statement allows you to define:

- whether the accounting or calculation phase of the UTM accounting is to be activated at the start of the UTM application,
- when an accounting record is written,
- the weighting with which resources are to be evaluated in the accounting phase.

If the ACCOUNT control statement is not specified, then this has the same effect as ACCOUNT ACC=NO. Only the first ACCOUNT statement of a KDCDEF run is evaluated.

UTM accounting can also be activated and deactivated via the administration, even if no ACCOUNT statement is issued in the KDCDEF generation. In this case the default values apply.

You may only specify the ACCOUNT statement once within a KDCDEF run.

> The UTM accounting functions and the format of accounting records written by openUTM are described in the openUTM manual “Using UTM Applications”.

ACCOUNT	ACC={ YES | NO | CALC } [,CPUUNIT=cpuunit] [,IOUNIT=iounit] [,MAXUNIT=maxunit] [,OUTUNIT=outunit]

ACC=
specifies which UTM accounting functions are to be executed. ACC is a mandatory operand.

YES
openUTM is to activate the accounting phase of UTM accounting after the application start.

NO
The accounting functions are not activated after the application start.
You can switch on the accounting functions during live operation using the administration command KDCAPPL ..., ACC=ON or via the program interface for administration (see the openUTM manual “Administering Applications”).

CALC
openUTM is to activate the calculation phase after the application start.

CPUUNIT=
cpuunit

specifies the weighting with which a CPU second is evaluated in the accounting phase of the UTM accounting. Fractions of a CPU second are billed proportionally.
You must enter an integer here.

Default value: 0
Minimum value: 0
Maximum value: 32767

IOUNIT= iounit

specifies the weighting with which 100 disk I/Os are evaluated in the accounting phase.

Fractions of 100 inputs/outputs are billed accordingly. You must enter an integer here.

Default value: 0
Minimum value: 0
Maximum value: 32767

> *Unix, Linux and Windows systems:*
> This operand is not used because these operating systems do not provide information on disk I/O.

MAXUNIT= maxunit

specifies the number of accounting units at which openUTM is to create an accounting record for a particular user (USER). You must enter an integer here.

Default value:
99 999 999 (=10^8-1) i.e. an accounting record is normally created only on connection shutdown.

Minimum value: 1
Maximum value: 99 999 999 (=10^8 - 1)

OUTUNIT= outunit

specifies the weighting with which a print job (FPUT NE) is evaluated for accounting purposes. You must enter an integer here.

Default value: 0
Minimum value: 0
Maximum value: 4095

6.5.4 APPLICATION-CONTEXT - define the application context

The APPLICATION-CONTEXT control statement is required only for communication based on the OSI TP protocol. You only have to specify the APPLICATION-CONTEXT statement if you want to define an additional application context.

It allows you to define the application context used for communication via OSI TP. The application context determines the rules governing data transfer between the
communication partners. It defines how the user data is encoded for transfer, and the format in which data is transferred. The application context must be coordinated with the partner.

The APPLICATION-CONTEXT statement enables you to define a local name for an application context, and to assign an object identifier and the abstract syntaxes belonging to this application context.

openUTM generates the standard application contexts UDTAC, UDTDISAC, XATMIAC, UDTCCR, UDTSEC and XATMICCR.

`APPLICATION-CONTEXT`	`application_context_name`
	`  ,OBJECT-IDENTIFIER=object_identifier`
	`  ,ABSTRACT-SYNTAX={ abstract_syntax_name |`
	`  (abstract_syntax_name,...) }`

application_context_name

A local name for an application context up to eight characters in length.

application_context_name must be unique within the UTM application.

OBJECT-IDENTIFIER=object_identifier

Object identifier of the application context specified as follows:

object_identifier=(number1,number2, ... ,number10)

number is a positive integer in the range 0 to 67108863. For object_identifier, you can specify two to ten integers enclosed in parentheses, each of which is separated by a comma. The number of integers entered and their positions are relevant.
Instead of the integer itself, you can also specify the symbolic name assigned to this integer. The table in section "OSI terms" shows the permitted values for number at the various positions.

object_identifier must be unique within the UTM application, i.e. another application context must not be generated with the same object identifier.

ABSTRACT-SYNTAX= Abstract syntax assigned to the application context for the transfer of user data.

abstract_syntax_name

Name of an abstract syntax defined using the ABSTRACT-SYNTAX control statement.

(abstract_syntax_name, ..., abstract_syntax_name)

List of up to nine abstract syntaxes separated by commas. Each abstract syntax specified in *abstract_syntax_name* must be defined beforehand using the ABSTRACT-SYNTAX statement.

The default UTM syntaxes CCR, UDT, XATMI, and UTMSEC need not be explicitly generated.

To work with transaction processing, a application context must be selected that contains the abstract syntax CCR.

If sign-on data is to be passed in a APRO call, then a application context must be selected that contains the abstract syntax UTMSEC.

If both partners use the XATMI interface, then a application context must be selected that contains the abstract syntax XATMI.

openUTM automatically generates the application contexts UDTAC, UDTDISAC, XATMIAC, UDTCCR, UDTSEC and XATMICCR, which are defined as follows:

Generation of “UDTAC”:

```
APPLICATION-CONTEXT UDTAC, -
OBJECT-IDENTIFIER=(1, 0, 10026, 6, 2), -
ABSTRACT-SYNTAX=UDT
```

Symbolic representation of the object identifier:

```
(iso, standard, tp, udt, application-context)
```

Generation of “UDTDISAC”:

```
APPLICATION-CONTEXT UDTDISAC, -
OBJECT-IDENTIFIER=(1, 0, 10026, 6, 2, 1), -
ABSTRACT-SYNTAX=UDT
```

Symbolic representation of the object identifier:

```
(iso, standard, tp, udt, application-context, with-tp)
```

Generation of “XATMIAC”:

```
APPLICATION-CONTEXT XATMIAC, -
OBJECT-IDENTIFIER=(1, 2, 826, 0, 1050, 4, 2, 1), -
ABSTRACT-SYNTAX=(XATMI)
```

Symbolic representation of the object identifier:

```
(iso, national-member-body, bsi, disc, xopen, xatmi, application-context, atpll-21-
31)
```

Generation of “UDTCCR”:

```
APPLICATION-CONTEXT UDTCCR, -
OBJECT-IDENTIFIER=(1, 0, 10026, 6, 2), -
ABSTRACT-SYNTAX=(UDT, CCR)
```

Symbolic representation of the object identifier:

```
(iso, standard, tp, udt, application-context)
```

Generation of “UDTSEC”:

```
APPLICATION-CONTEXT UDTSEC, -
OBJECT-IDENTIFIER=(1, 3, 0012, 2, 1107, 1, 6, 1, 3, 0), -
ABSTRACT-SYNTAX=(UDT, UTMSEC, CCR)
```

Symbolic representation of the object identifier:

```
(iso, identified-organisation, icd-ecma, member-company, siemens-units, sni,
transaction-processing, utm-security, application-context, version)
```

Generation of “XATMICCR”:

```
APPLICATION-CONTEXT XATMICCR, -
OBJECT-IDENTIFIER=(1, 2, 826, 0, 1050, 4, 2, 1), -
ABSTRACT-SYNTAX=(XATMI, CCR)
```

Symbolic representation of the object identifier:

```
(iso, national-member-body, bsi, disc, xopen, xatmi, application-context, atpll-21-
31)
```

6.5.5 AREA - define additional data areas

The AREA statement allows you to define the name, properties, and sequence of additional shareable data areas. The structure of these areas is not defined by openUTM and can be defined as chosen. The addresses of such areas are passed to the program unit as parameters at the start of the program with the address of the communication area and the standard primary working area.

> ¡ You have an alternative to administering areas with openUTM using AREA statements in most programming languages (especially under COBOL and C/C++). The alternative is to declare areas as external data areas and to access these areas from the program units. This option offers a number of benefits compared with AREAs. You will find more information on this subject in the openUTM manual „Programming Applications with KDCS”.

Each area to be defined in openUTM must be defined in a separate AREA statement. The order of the AREA statements indicates the order in which these areas must be specified in the parameter list and declared in the program unit (e.g. on BS2000 systems in the LINKAGE-SECTION under COBOL). If the area defined on the *n*-th location is required, then all areas in the parameter list and in the data declaration must be specified or declared up to this area.

It is possible to specify up to 99 AREA statements in a single generation run.

> ¡ AREAs in UTM cluster applications are local to the node, i.e. each node application has its own instance of each AREA.

Generating areas on BS2000 systems

On BS2000 systems areas can be created:

- in the global common memory pool (for all applications).
- in the local common memory pool (for all application processes started under the same user ID).
- in non-privileged subsystems.
- in the linked application program.

The following applies for the AREA statement:

- If you specify the operand LOAD-MODULE=, you must also write a LOAD-MODULE statement. Note that no load module may be referenced which has been generated with LOAD-MODULE ...LOAD-MODE=ONCALL.
- AREA statements that do not contain the LOAD-MODULE operand define data areas that are linked statically to the application program.
- The default values for AREA are set using the DEFAULT PROGRAM statement.

AREA	areaname [,LOAD-MODULE=lmodname]

areaname Name of the area. *areaname* is an alphanumeric value up to 32 characters in length. *areaname* must be a module.

LOAD-MODULE=lmodname

lmodname can be up to 32 characters in length.

LOAD-MODULE= identifies the name of the load module to which the module is linked. This load module must be defined using the LOAD- MODULE statement, and must not be generated with the operand LOAD-MODE=ONCALL.

Generating areas on Unix, Linux and Windows systems

An area must be explicitly defined, compiled, and linked to the program unit as external C/C++ data structures.

In the AREA statement, you can define whether the area is transferred directly to the program unit, or is accessed indirectly by means of a pointer. When accessing indirectly, a pointer must be supplied with the address of the area before the first program unit is started. You can set the addresses before compiling or during the application run in the event exit START, for example.

AREA	areaname [,ACCESS={ DIRECT | INDIRECT }]

areaname Name of the area. *areaname* is an alphanumeric value up to 32 characters in length. *areaname* must be a module.

ACCESS= Mode of access to the additional data area

DIRECT The area is defined directly as a C data structure.Default: DIRECT

INDIRECT The area is defined as a pointer. The pointer *areaname* must be supplied with the address of the area. It is possible to first set the address during the application run, e.g. you can store the address of a shared memory area in the pointer in the START event exit.

6.5.6 BCAMAPPL - define additional application names

The BCAMAPPL statement allows you to assign additional application names to the UTM application for client /server communication and distributed processing via LU6.1. Each application name defines a transport system endpoint that can be used to establish connections to the UTM application.

The primary application name of the UTM application is specified in the APPLINAME operand of the MAX statement. Please note the following:

- *BS2000 systems:*
  You may issue the BCAMAPPL statement only for additional BCAM names of the application. You must not issue a BCAMAPPL statement for the primary application name.
- *Unix, Linux and Windows systems:*
  - You will also need to issue a BCAMAPPL statement for the primary application name if you also wish to establish connections via this name to partner applications or clients.
  - Only a maximum of 1000 connections can be established at a time per application name. If more concurrent connections are needed by your application, you must define more than one application name.

The BCAMAPPL statement can be issued several times. However, to ensure that resources are not unnecessarily occupied, you should only generate as many BCAMAPPL statements (i.e. application names) as are necessary.

It is necessary to generate additional application names for your UTM application if:

- parallel connections via LU6.1 are to be defined to other applications (distributed processing). In this event, additional application names must be generated in at least one of the applications involved.
- communication with a partner is to be done via the socket interface (native TCP/IP). You will need a separate BCAMAPPL name (with T-PROT=SOCKET) for the communication via the socket interface. This name cannot be used for communication via other transport protocols.
- you select the transport protocol (not NEA) for a partner of a UTM application generated with PTYPE=APPLI, PTYPE=UPIC-R, or generated as a partner of a LU6.1 application.
- you establish multiplex connections to a partner of a UTM application.
- you want to communicate with a UTM application on Unix or Windows systems.
- you want to establish connections via the RFC1006 protocol. In this case you must define a separate BCAMAPPL name for the communication via RFC1006.

> In all K-messages which contain the UTM application name as an insert, the name defined in the APPLINAME operand of the MAX statement will be displayed.

BCAMAPPL statement on BS2000 systems

```
BCAMAPPL  appliname
            ,LISTENER-PORT=number only allowed and mandatory for T-PROT=SOCKET
          [ ,SIGNON-TAC={ *NONE | tacname } ]
                        [ ,T-PROT={ NEA | ISO | RFC1006 | SOCKET |
                                    (SOCKET [,*ANY | *USP | *HTTP] [,SECURE])} ]

          [ ,USER-AUTH = *NONE | *BASIC ]
```

appliname

Additional BCAM name of the UTM application. *appliname* can be up to eight characters in length. *appliname* must not be identical to the application name you specified in MAX ..., APPLINAME= or in ACCESS-POINT..., TRANSPORT-SELECTOR=.

appliname must be unique in the local system for each host.

LISTENER-PORT= *number*

Only permitted for T-PROT=SOCKET. In this case, it is mandatory to specify the LISTENER-PORT.

LISTENER-PORT specifies the port number on which openUTM waits for external connection establishment requests.

If LISTENER-PORT is generated together with T-PROT=(SOCKET,..., SECURE), then in addition to the port number defined with *number*, the port number *number*+1 is also assigned by the UTM application.
The second port number is required for communication between the UTM application and the reverse proxy.

All port numbers between 1 and 65535 are allowed.

Each port number may only be used **once** in the local system. It may be when starting openUTM that some port numbers are already reserved by the system or other TCP/IP applications, or that privileged port numbers may not be used. In this case, the start of the UTM application is aborted.

SIGNON-TAC =

Specifies whether a sign-on service is to be started for connections that are established using the application names *appliname* (=transport system access point). If a sign-on service is to be started, you must specify the name of the transaction code via which the sign-on service is to be started.

*NONE

For connections to the UTM application that are to be established using the application name *appliname*, no sign-on service is to be started, regardless of whether the TAC is generated with KDCSGNTC or not.

If the T-PROT=(SOCKET,*ANY | *HTTP) is given for the BCAMAPPL, then the value *NONE has to be specified for SIGNON-TAC, if the TAC KDCSGNTC is generated for the application.

tacname

Name of the service TAC started via the sign-on service.

The transaction code *tacname* must be generated using a TAC statement. In the TAC statement, you must not modify the following default settings for the transaction code:

- API = KDCS,
- CALL = FIRST or BOTH,
- ENCRYPTION-LEVEL = NONE,
- PGWT = NO,
- TACCLASS = 0,
- TYPE = D,
- no limitation on data access authorizations, i.e. the operands ACCESS-LIST and LOCK may not be specified

For UPIC partners, the sign-on service is only started if UPIC=YES is generated in the SIGNON statement. In the case of UPIC partners, the signon service is not started when the connection is established. Instead, it is started before a UPIC conversation is started (see also SIGNON statement, OMIT-UPIC-SIGNOFF= parameter in section "SIGNON - control the sign-on procedure").
For LU6.1 partners, no sign-on service is started.

tacname may not be assigned to a program (PROGRAM operand of a TAC statement) that is located in a load module generated with LOAD-MODE=ONCALL.

Default:

- KDCSGNTC as far as it is generated in the application (KDCSGNTC = standard sign-on service; generated with a TAC statement)
- otherwise *NONE

> **CAUTION!**
> Those communication partners that establish their connection to the UTM application via the primary application name (generated in MAX APPLINAME=) can only have a sign-on service that is generated using the transaction code KDCSGNTAC.

T-PROT=

Transport protocols to be used on the connections to partner applications that are established through this application name.

NEA

An NEA transport protocol is used.

Default: NEA

ISO — An ISO transport protocol is used.

Whether or not an ISO transport connection can be established to this application and which transport protocol will actually be used depends on the generation of the transport system. As parallel connections are allowed for ISO transport connections although they are not supported by openUTM, openUTM accepts the connection of the contention winner (CON) or of the partner with the alphabetically smaller name pair (*ptermname*, processor name, PTERM statement) in case of a contention.

RFC1006 — TCP/IP is used with the RFC1006 convergence protocol.
RFC1006 is synonymous with T-PROT=ISO on BS2000 systems.

T-PROT=RFC1006 or T-PROT=ISO must be used for communication with openUTM on Unix, Linux or Windows systems.

SOCKET — Native TCP/IP is to be used as the transport protocol, i.e. communication is to be handled via the socket interface.

If you specify T-PROT=SOCKET, then you must define a port number in the LISTENER-PORT operand.

You will find more information on SOCKET in the section "Providing address information".

With sub-parameters the type of socket protocol can be speciified that is to be used on these connections.

*USP — The UTM socket protocol shall be used on connections of this transport system access point.
*USP is the default value.

*HTTP — The HTTP protocol shall be used on connections of this transport system access point.

*ANY — On connections of this communication end point both of the UTM socket protocol and the HTTP protocol are supported.
WIth this configuration the protocol of a connection is determined by the protocol of the first message received.

*SECURE

When SECURE is specified as additional sub-parameter, then the TLS functionality of the secure socket layer is used for all communication on these connections.

If SECURE is specified for an application in BS2000 systems, UTM starts an additional ENTER process with a reverse proxy for this application. The purpose of this proxy process is to accept SSL connections for the UTM application and pass them on to the UTM application.

UTM passes a maximum of three listener port numbers to the reverse proxy process. This means that for a UTM(BS2000) application with a maximum of three BCAMAPPL statements, the attribute SECURE should be specified. If more than three BCAMAPPL statements with the SECURE attribute are required for an application, a user must create the start procedure of the reverse proxy process himself and start the process manually.

The prerequisite for starting the reverse proxy process is that a job variable with the base name of KDCFILE is cataloged when the application is started. For details see the description for the use of such a job variable and of the reverse proxy process in generell in the manual "Using UTM Applications in BS2000 Systems" in the chapter "Starting UTM Applications".

USER-AUTH =

The USER-AUTH parameter specifies which authentication mechanism HTTP-clients must use for this application.

The value that is set here for this parameter applies to all HTTP messages that are received using this application name and whose path specification is not mapped to a TAC using an HTTP-DESCRIPTOR statement. For the latter cases, the parameter USER-AUTH acts on the HTTP-DESCRIPTOR statement.

If the UTM application is generated without users, only the value *NONE may be specified for USER-AUTH.

*BASIC

The Basic Authentication Scheme from RFC 2617 is to be used to transfer authentication data. If Basic Authorization is required for an HTTP request, but no Authorization Header is contained in the HTTP request, UTM requests authentication data using a response with status code 401 Unauthorized.

*NONE

If *NONE is specified, the client does not have to pass any authentication data. UTM uses the connection user for such a request if the client does not automatically send authentication information in the HTTP request.

Default: *NONE

BCAMAPPL statement on Unix, Linux and Windows systems

On Unix, Linux and Windows systems the operands *appliname*, LISTENER-PORT (TCP/IP port number), T-PROT (transport protocols used) and TSEL-FORMAT (format identifier) are used to specify the address.

BCAMAPPL	appliname [,LISTENER-ID=number] [,LISTENER-PORT=number] [,SIGNON-TAC={ *NONE | tacname }] [,T-PROT={ RFC1006 | SOCKET | (SOCKET [,*ANY | *USP | *HTTP] [,SECURE])}] [,USER-AUTH = *NONE | *BASIC]] [,TSEL-FORMAT={ T | E | A }]

appliname

Additional name of the UTM application. *appliname* can be up to eight characters in length. *appliname* must not be identical to the application name you specified in ACCESS-POINT..., TRANSPORT-SELECTOR=.

In addition, the name must be different from the application name that you specified in the BCAMAPPL operand of the CLUSTER statement.

appliname must be unique throughout the network.

KDCDEF creates a T-selectors from *appliname* for the transport system. The T-selectors is part of the transport address of the application that is used to address the application from partner applications when establishing a connection.

Exception:
appliname is only relevant internally in UTM for T-PROT=SOCKET, e.g. for the administration. The name only needs to be unique within the application.

LISTENER-ID= number

This assigns a listener ID to the application name as administrative information.

Listener IDs can be specified for application names and access points. Further information can be found in the description of the ACCESS-POINT statement.

You can use the listener IDs to distribute the network connections of the access points to different network processes. All connections of an access point are managed by the same network process.

BCAMAPPL names with T-PROT=SOCKET (communication via the socket interface) comprise a separate set of numbers, i.e. the access points for communication via the socket interface are managed using different network processes than the access points for other transport protocols.

If you do not specify a listener ID, then openUTM assigns the value 0 as the listener ID. All connections without a listener ID that are not established via the socket interface are combined into a single network process, and all connections without a listener ID that are established via the socket interface are combined into another single network process.

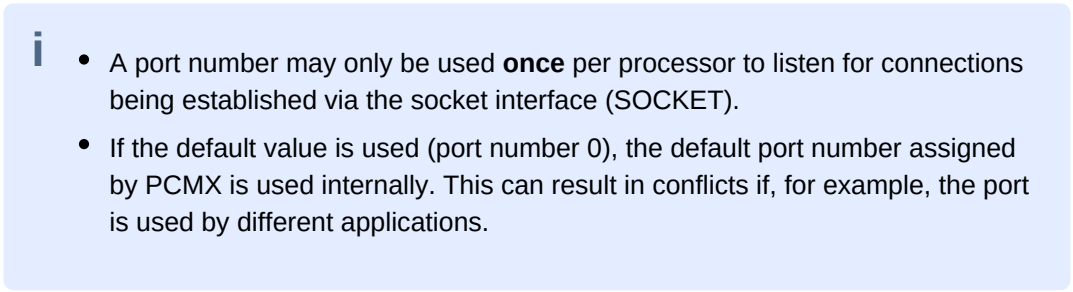
Default value: 0
Minimum value: 0
Maximum value: 65535

LISTENER-PORT= number

Port number of the UTM application.

All port numbers between 1 and 65535 are allowed.

With T-PROT=RFC1006 and OPTION CHECK-RFC1006=YES and withT-PROT=SOCKET, a port number must be specified for LISTENER-PORT. In all other cases, the default value is 0 (no port number).



- A port number may only be used **once** per processor to listen for connections being established via the socket interface (SOCKET).
- If the default value is used (port number 0), the default port number assigned by PCMX is used internally. This can result in conflicts if, for example, the port is used by different applications.

SIGNON-TAC = Specifies whether a sign-on service is to be started for connections that are established using the application names *appliname* (=transport system access point). If a sign-on service is to be started, you must specify the name of the transaction code via which the sign-on service is to be started.

*NONE

For connections to the UTM application that are to be established using the application name *appliname*, no sign-on service is to be started, regardless of whether the TAC is generated with KDCSGNTC or not.

If the T-PROT=(SOCKET,*ANY | *HTTP) is given for the BCAMAPPL, then the value *NONE has to be specified for SIGNON-TAC, if the TAC KDCSGNTC is generated for the application.

tacname

Name of the service TAC started via the sign-on service.

The transaction code *tacname* must be generated using a TAC statement. In the TAC statement, you must not modify the following default settings for the transaction code:

- API = KDCS,
- CALL = FIRST or BOTH,
- ENCRYPTION-LEVEL = NONE,
- PGWT = NO,
- TACCLASS = 0,
- TYPE = D,
- no limitation on data access authorizations, or in other words, the operands ACCESS-LIST and LOCK may not be specified

For UPIC partners, the sign-on service is only started if UPIC=YES is also generated in the SIGNON statement. In the case of UPIC partners, the signon service is not started when the connection is established. Instead, it is started before a UPIC conversation is started (see also SIGNON statement, OMIT-UPIC-SIGNOFF= parameter in section "SIGNON - control the sign-on procedure"). For LU6.1 or OSI TP partners, no sign-on service is started.

Default:

- KDCSGNTC as far as it is generated in the application (KDCSGNTC = standard sign-on service; generated with a TAC statement)
- otherwise *NONE

> **CAUTION!**
> If the application name specified in *appliname* corresponds to the primary application name (generated in MAX APPLINAME=) then for SIGNON-TAC= you may specify KDCSGNTC (standard sign-on service), *NONE or leave it blank.

T-PROT=

Address formats of the T-selectors in the transport address.

You can specify the following address formats for T-PROT.

RFC1006

Address format RFC1006

You will find more information on the RFC1006 address format in the section “PCMX documentation” (openUTM documentation).

Default: RFC1006

SOCKET — Communication is done via the socket interface.
No other address specifications are required other than T-PROT=SOCKET, LISTENER-PORT and *appliname*.
You will find more information on SOCKET in the section "Providing addressinformation".

With sub-parameters the type of socket protocol can be speciified that is to be used on these connections.

*USP — The UTM socket protocol shall be used on connections of this communication end point.

*USP is the default value.

*HTTP — The HTTP protocol shall be used on connections of this communication end point.

*ANY — On connections of this communication end point both of the UTM socket protocol and the HTTP protocol are supported.
WIth this configuration the protocol of a connection is determined by the protocol of the first message received.

*SECURE — If SECURE is also specified as an additional sub-parameter, then communication on these connections takes place using TLS via the Secure Socket Layer.

USER-AUTH = — The USER-AUTH parameter specifies which authentication mechanism clients must use for this application.

The value that is set here for this parameter applies to all HTTP messages that are received using this application name and whose path specification is not mapped to a TAC using an HTTP-DESCRIPTOR statement. For the latter cases, the parameter USER-AUTH acts on the HTTP-DESCRIPTOR statement.

If the UTM application is generated without users, only the value *NONE may be specified for USER-AUTH.

*BASIC — The Basic Authentication Scheme from RFC 2617 is to be used to transfer authentication data. If Basic Authorization is required for an HTTP request, but no Authorization Header is contained in the HTTP request, UTM requests authentication data using a response with status code 401 Unauthorized.

*NONE — If *NONE is specified, the client does not have to pass any authentication data. UTM uses the connection user for such a request if the client does not automatically send authentication information in the HTTP request.

Default: *NONE

TSEL-FORMAT= — Format identifier of the T-selectors to be created from *appliname*.

The format identifier specifies the encoding of the T-selector in the transport protocol. You will find more information in the section “PCMX documentation” (openUTM documentation).

T — TRANSDATA format (coding in EBCDIC)
In this case *appliname* must be exactly 8 characters long and must not include lowercase letters.

E — EBCDIC format

A ASCII format

Default:

T if the character set of *appliname* matches to the TRANSDATA format
E in all other cases

It is recommended for TNS-less operation via RFC1006 to explicitly specify a value for TSEL-FORMAT.

6.5.7 CHAR-SET- assign names to code tables (BS2000 systems)

This statement is only needed when the application communicates with HTTP-Clients.

Messages from HTTP-Clients are normally encoded in an ASCII character set. On the other hand BS2000 systems use EBCDIC character sets. The character set of the payload (message body) of an HTTP message can be specified in the Content-Type header of the HTTP protocol. UTM can perform a code conversion of the payload, if the character set name given in the Content-Type header of the HTTP message matches exactly one of the names generated with the CHAR-SET statement. Note that the names are interpreted case-insensitive.

Each of the four code conversion tables of UTM may be assigned up to four character-set names.

`CHAR-SET`	`{ SYS1 | SYS2 | SYS3 | SYS4 }` `, NAME= (C'char-set-name', ...)`

SYS1 | SYS2 | SYS3 | SYS4

With this parameter one of the four code conversion tables of UTM is selected. For further information about code conversion in UTM applications see chapter "Code conversion".

NAME= (C'char-set-name', ...)

With the parameter NAME up to four character set names may be defined, which will be assigned to the code conversion table specified in this statement. The spelling of the names is case-insensitive. The maximum length of the name is 32 characters.

The defined character set names must be unique within the application. At most four character set names may be assigned to one code conversion table. This may be done by one or more CHAR-SET statements.

6.5.8 CLUSTER - define global properties of a UTM cluster application (Unix, Linux and Windows systems)

The CLUSTER statement is used to configure a UTM cluster application. The operands of the CLUSTER control statement can be split over several CLUSTER statements.

If you specify the same operand in several CLUSTER statements, the first specification is taken as the valid one. No message is issued.

If a cluster statement is specified, you must also specify at least two CLUSTER-NODE statements. If a CLUSTER statement is specified, KDCDEF implicitly generates a BCAMAPPL entry with the BCAMAPPL name specified in the CLUSTER statement.

The effect of the CLUSTER statement depends on the value specified in the OPTION statement, see section "OPTION - manage the KDCDEF run".

If you change specifications in the CLUSTER statement or the CLUSTER-NODE statements with a new generation, you must create new UTM cluster files and a new KDCFILE (OPTION GEN=CLUSTER, KDCFILE) and use this file to apply the changes.
Exception:
The size of the cluster page pool can be increased during operation, i.e. without it being necessary to generate new UTM cluster files. When this is done, the number of cluster page pool files must not be changed.

CLUSTER	CLUSTER-FILEBASE = cluster_filebase
	  ,BCAMAPPL = cluster_applname
	  ,LISTENER-PORT = port_number
	  ,USER-FILEBASE = user_filebase
	[,ABORT-BOUND-SERVICE = { NO | YES }]
	[,CHECK-ALIVE-TIMER-SEC = time]
	[,COMMUNICATION-REPLY-TIMER-SEC = time]
	[,COMMUNICATION-RETRY-NUMBER = number]
	[,DEADLOCK-PREVENTION = { NO | YES }]
	[,EMERGENCY-CMD = command_string1]
	[,FAILURE-CMD = command_string2]
	[,FILE-LOCK-RETRY = number]
	[,FILE-LOCK-TIMER-SEC = time]
	[,PGPOOL=(number,warnlevel)]
	[,PGPOOLFS=number]
	[,RESTART-TIMER-SEC = time]
	[,LISTENER-ID=number]

The operands CLUSTER-FILEBASE, BCAMAPPL, LISTENER-PORT and USER-FILEBASE must always be specified. The specifications in the OPTION statement determine whether and in what way these operands are then evaluated.

CLUSTER-FILEBASE=cluster_filebase

Name prefix or directory for the UTM cluster files. Some of the UTM cluster files are generated by KDCDEF (see list below) while others are not generated until runtime.

The operand CLUSTER-FILEBASE is only evaluated if GEN=CLUSTER or GEN=(CLUSTER,...) is specified in the OPTION statement. In this case, KDCDEF generates the following files:

- the cluster configuration file
- the cluster user file
- the cluster page pool files.
- the cluster GSSB file
- the cluster ULS file

In this case, these files must not already exist.

Mandatory operand.

cluster_filebase defines the directory in which the UTM cluster files are to be stored. The directory must be created before the KDCDEF run.

The UTM cluster files are created under the file names `UTM-C.xxxx`, where`xxxx` is file-specific, see "UTM cluster files".

The files can be copied to a different directory to operate the UTM cluster application. Specify the name which is then valid in the start parameters when the application is started. The name can be up to 42 characters in length and must comply with the syntax for file names.

BCAMAPPL= cluster_applname

Name of the communication end point for cluster-internal communication.

The name specified here must be different from the names specified for TRANSPORT-SELECTOR under MAX APPLINAME, in other BCAMAPPL statements or in ACCESS-POINT statements. In addition, the name specified here must not be used by other applications on the computers of the UTM cluster application as the name of a communication end point.

The name generated here must not be referenced in other statements (e.g. in the PTERM statement) as the BCAMAPPL name.

The name can be up to 8 characters in length.

Mandatory operand.

LISTENER-PORT= port_number

Port number for cluster-internal communication.

This operand specifies the port number on which the local application listens for external connection requests.

Enter any port number between 1 and 65535.

Note that the port number specified here must not be used anywhere else on the computers of the UTM cluster. The port number must also differ from the other port numbers used by this application. KDCDEF does not, however, check this.

Mandatory operand.

USER-FILEBASE= user_filebase

Name prefix or directory for the current cluster user file of a UTM cluster application. The operand USER-FILEBASE is only evaluated if GEN=KDCFILE, GEN=(KDCFILE, ROOTSRC) or GEN=ROOTSRC is specified in the OPTION statement.

- If GEN=KDCFILE or GEN=(KDCFILE,ROOTSRC), the cluster user file must exist under the name taken from *user_filebase*. KDCDEF evaluates the file and extends it if necessary. The cluster user file can already be open for the KDCDEF run of a running UTM cluster application.
- If GEN=ROOTSRC then the cluster user file may already exist but this is not mandatory. If it does exist then it is checked but not modified.

Mandatory operand.

The name can be up to 42 characters in length and must comply with the syntax for file names.

ABORT-BOUND-SERVICE

This parameter determines how UTM behaves when a user who has an open service in a node application signs on.

NO If there is a node-bound service for a user on sign-on (see note), then the user can only sign on at the node application to which the open service is bound; Sign-on attempts at any other node application are rejected.

Default in UTM-S applications.

This value is not permitted in UTM-F applications.

YES

If when a user signs on at a node application, there is a node-bound service for this user that is bound to another node application that has been terminated, then the user is able to sign on provided that no transaction of the open service has the state PTC. No service restart is performed

The open service is terminated abnormally the next time the node application to which it is bound is started.

Default in UTM-F applications.

> A service is node-bound if it
>
> - has a job-receiver service
> - or is an inserted service resulting from service stacking.
>
> In addition, a service associated with a user is node-bound as long as the user is signed-on at a node application.

CHECK-ALIVE-TIMER-SEC=time

Interval in seconds at which a node application of a UTM cluster application checks the availability of another node application.

Minimum value: 30
Maximum value: 3600
Default value: 600

COMMUNICATION-REPLY-TIMER-SEC=time

Time in seconds that a node application of a UTM cluster application waits for a response after sending a message to another node application.

If there is no response in the time specified here, it must be assumed that the other node application has failed. If you have selected a value greater than zero for COMMUNICATION-RETRY-NUMBER, it is only assumed that the other node application has failed after the number of retry attempts has been reached.

Minimum value: 1
Maximum value: 60
Default value: 10

COMMUNICATION-RETRY-NUMBER=number

Number of retry attempts to establish communication with another node application if this node application does not respond within the time specified under COMMUNICATION-REPLY-TIMER. If the monitored node application also fails to respond to any of the retry attempts, it is flagged as failed.

Minimum value: 0, i.e. no retry after a timeout.
Maximum value: 10
Default value: 1

DEADLOCK-PREVENTION=

In UTM cluster applications, information concerning locked data areas (GSSB, TLS, ULS) is stored in a file. Before a service waits at a locked data area, UTM can check whether the new wait situation might result in a deadlock. To do this, additional file I/Os are necessary.

This parameter specifies whether or not UTM performs additional checks in order to prevent deadlocks.

YES — UTM performs additional checks of the GSSB, TLS and ULS data areas in order to prevent deadlocks.

NO — UTM does not perform any additional checks of the GSSB, TLS and ULS data areas in order to prevent deadlocks If a deadlock occurs in one of these data areas then this is resolved by means of a timeout. See also MAX statement, operand RESWAIT=time1 ("MAX - define UTM application parameters").

Default: NO

In productive operation, it is advisable to set this parameter to YES only if timeouts occur frequently when accessing these data areas.

EMERGENCY-CMD= command_string1

Name of a script.

This operand passes a command string containing a command to be executed.

The emergency script is called by openUTM if a failed node application was not restarted after the failure script has been called and the restart timer (RESTART-TIMER-SEC parameter) has expired.

The emergency script can be used, for example, to restart the failed computer in a cluster or perform a node recovery for a failed node application.

The emergency script is always executed on the computer of the monitoring node application.

The name passed here is not parsed by KDCDEF.

command_string1 can be up to 200 characters in length. The way in which the emergency script is specified depends on the operating system.

> When openUTM is installed, platform specific templates are supplied with the name `utm-c.emergency`.

Unix and Linux systems:

Specify the fully qualified name of a Unix Sshell script.

Example: `EMERGENCY-CMD = '` *utmpath* `/shsc/utm-c.emergency'`

Windows systems:

Specify the fully qualified name of a Windows command script.

Example: `EMERGENCY-CMD = '` *utmpath*`\shsc\utm-c.emergency.cmd'`

Calling the script command_string1 during an application run

Six arguments are passed to the script *command_string1*. These identify the failed cluster node and allow corrective measures to be initiated.

The arguments are passed in the following sequence:

1st argument	Name of the UTM application
2nd argument	Filebase name of the KDCFILE of the failed node application
3rd argument	Host name of the failed node
4th argument	Virtual host name of the failed node
5th argument	Reference name of the failed node application (NODE-NAME parameter in the CLUSTER-NODE statement)
6th argument	Term Application Reason: Error code in the UTM dump of the failed node, see message K060 in openUTM manual ”Messages, Debugging and Diagnostics”. You can decide whether or not to restart the node application on the basis of this error code. • The error code ASIS99 means that the node was terminated abnormally by the administrator with KDCSHUT KILL and that it should not normally be restarted. • In the case of all other error codes (with the exception of ENDPET), the node application was terminated abnormally and should normally be restarted. • The error code ENDPET means that the node application was terminated normally by the administrator with KDCSHUT even though there was at least one distributed transaction in the PTC state (prepare to commit). In this case the node application should be restarted if possible in order to resolve the PTC state and release any locks in the node application or a partner application.

The return code of the procedure or script is not evaluated.

Unix and Linux systems:

The generated script *command_string1* is started as a background process.

It is called with the six arguments listed above.

Windows systems:

The command script *command_string1* is called with the Windows command START without waiting for it to terminate.

It is called with the six arguments listed above.

FAILURE-CMD= command-string2

Name of a script.

command_string2 can be up to 200 characters in length. The way in which the failure script is specified depends on the operating system.

The failure script is called by openUTM if a node application terminates abnormally or if failure of a node application is detected. A user can use the failure script to restart the failed node application, for instance.

The failure script is always executed on the computer of the monitoring node application.

Otherwise, the syntax and call method for FAILURE-CMD are identical to the syntax and call method of "EMERGENCY-CMD" (see "CLUSTER - Define global properties of a UTM cluster application").

> **i** When openUTM is installed, platform specific templates are supplied with the name `utm-c.failure`.

FILE-LOCK-RETRY= number

Number of retries for a lock request for a file that is global to the cluster if the lock was not assigned in the time specified in FILE-LOCK-TIMER-SEC.

Minimum value: 1
Maximum value: 10
Default value: 1

FILE-LOCK-TIMER-SEC= time

Maximum time in seconds that a node application of a UTM cluster application waits for a lock to be assigned to a file that is global to the cluster.

Minimum value: 10
Maximum value: 60
Default value: 30

LISTENER-ID= number

This parameter is used to select a network process for internal cluster communication.

Minimum value: 0
Maximum value: 65535
Default value: 0

PGPOOL= (number, warnlevel)

Specifies the size of the cluster page pool and the warning level for cluster page pool utilization. The cluster page pool stores the GSSB, ULS and the service data of users (USER statement) who are generated with RESTART=YES.

The cluster page pool can be extended during cluster operation while leaving the number of files unchanged, see the applicable openUTM manual “Using UTM Applications”.

number

Size of the cluster page pool in UTM pages.

For each generated node, at least 500 UTM pages are needed in the cluster page pool. The size of a UTM page is defined in the BLKSIZE operand of the MAX statement.

Default: 10,000 or the minimum size
Minimum value: 500 * number of cluster nodes
Maximum value: 16777215 - (2 * *number* in CLUSTER PGPOOLFS)

If the value specified here is smaller than the minimum value that UTM calculates from the number of generated nodes and the length generated in MAX RECBUF=*length*, then UTM increases *number* to the minimum size.

warning level

Percentage value specifying the cluster page pool utilization level at which a warning (message K041) is output.

Default: 80
Minimum value: 60
Maximum value: 99

Please note that the messages indicating that cluster page pool utilization has risen above or fallen below the warning level are only output for the node application that triggers the associated change in state. In contrast, all running node applications are affected by a potential cluster page pool bottleneck.

PGPOOLFS=

number

Number of files over which the user data is to be distributed in the cluster page pool.

The cluster page pool files are created using the cluster filebase that is specified in the CLUSTER-FILEBASE operand. They are given the suffixes CP01, CP02, ... CP10.

In addition, KDCDEF always creates a file with the suffix CPMD which is used to manage the cluster page pool and does not contain any user data.

Default: 1
Minimum value: 1
Maximum value: 10

RESTART-TIMER-SEC=

time

Maximum time in seconds that a node application requires for a warm start after a failure.

After a failure has been detected and the failure command for a failed node application has been called, the monitoring node application starts a timer with the time specified here. If the failed node application is not available after this time has expired, the emergency command is started for the failed node application.

If a value of 0 is specified, no timer is set for monitoring the restart of the failed node application.

Minimum value: 0, i.e. restart of the application is not monitored.
Maximum value: 3600
Default value: 0

6.5.9 CLUSTER-NODE - define a node application of a UTM cluster application (Unix, Linux and Window systems)

You use the CLUSTER-NODE statement to configure a node application of a UTM cluster application.

You can start up to 32 node applications simultaneously.

You are allowed to specify the CLUSTER-NODE statement up to 32 times for each UTM cluster application.

You must specify at least two CLUSTER-NODE statements if you want to generate a UTM cluster application. A CLUSTER statement must also be generated if you have specified a CLUSTER-NODE statement.

> ¡ If you change specifications in the CLUSTER statement or the CLUSTER-NODE statements with a new generation, you must create a new cluster configuration file (OPTION GEN=CLUSTER) and use this file to apply the changes.

`CLUSTER-NODE`	`  FILEBASE = node_filebase`
	`  ,HOSTNAME = host_name`
	`[ ,NODE-NAME = node_name]`
	`[ ,VIRTUAL-HOST = virtual_host_name]`

FILEBASE = node_filebase

Base name of the KDCFILE, the user log file and the system log file SYSLOG for this node application. When a node application is started, the UTM system files are expected under the name specified here. The KDCFILE must be accessible from all node applications.

This operand replaces the FILEBASE start parameter in a standalone UTM application.

The base names of the CLUSTER-NODE statements must differ from each other. The same restrictions apply as for MAX KDCFILE=*filebase*.

Mandatory operand.

node_filebase identifies the directory containing the KDCFILE and all the files of the application when a node application of a UTM cluster application is started. The name **specified here** must identify the same directory **from the perspective of all the computers of the cluster**. The name can be up to 27 characters in length.

HOSTNAME= host_name

Host name of this node. Specify the primary name of this host.

The name can be up to 64 characters in length.

The host names of the CLUSTER-NODE statements must differ from each other. Host names that only differ in terms of case are regarded as identical.

In the case of Unix and Linux systems, you must specify the name of the computer that is output by the command *uname -n*.

On Windows systems, you must specify the name of the computer that is entered in the Control Panel.

No distinction is made between uppercase and lowercase notation; KDCDEF always converts the host names into uppercase.

Mandatory operand.

NODE-NAME= node_name

Defines a reference name for the node application. This name can be used when configuring LU6.1 sessions as well as for node recoveries:

- Configuring LU6.1 sessions:
  The reference name defined here can be specified in the NODE-NAME parameter of an LSES statement in order to assign the LU6.1 session unambiguously to a node application. This enables openUTM to select the "right" session when establishing a session with a partner application.
- Node recovery:
  If a node recovery is to be performed for the node application generated here then the reference name defined here must be specified in the NODE-TO-RECOVER start parameter. For more details, consult "node recovery" in the openUTM manual “Using UTM Applications”.

Default value: NODE*nn*
nn = 01..32, where *nn* is determined by the sequence of CLUSTER-NODE statements during generation.

VIRTUAL-HOST= virtual_host_name

Has the same function as the MAX HOSTNAME parameter with UTM cluster applications. You are not allowed to specify the MAX HOSTNAME parameter in UTM cluster applications.

VIRTUAL-HOST allows the sender address for network connections established from this node application to be specified.

The name can be up to 64characters in length.

Default: blanks. This means that the default sender address of the transport system is used when connections are established.

This function is required in a cluster if the relocatable IP address is to be used as the sender address instead of the static IP address when establishing a connection.

No distinction is made between uppercase and lowercase notation; KDCDEF always converts the virtual host names into uppercase.

6.5.10 CON - define a connection for distributed processing based on LU6.1

The CON statement allows you to define a transport connection between the local UTM application and a partner application. It also assigns an LPAP partner to the real partner application, i.e. the logical access point of the partner application in the local application. You must define the LPAP partner in an LPAP statement (see "LPAP - define an LPAP partner for distributed processing based on LU6.1").
By issuing several CON statement for the same partner application, you can also define parallel transport connections.

Generation when standalone UTM applications are to be linked to UTM cluster applications

For more information on generating LU6.1 connections see "Distributed processing via the LU6.1 protocol"

When generating the CON, PTERM and MUX statements, please note that the name triplet (*appliname* or *ptermname*, *processorname*, *local_appliname*) must be unique within the generation run.

Example

If a PTERM statement has already been generated with

`PTERM` *partner_name1*`,PRONAM=` *processorname1*,

you cannot generate a CON statement with

`CON` *partner_name1* `PRONAM=` *processorname1*,

but you can enter

`CON` *partner_name1* `,PRONAM=` *processorname1* `,BCAMAPPL=` *local_appliname1*,

provided *local_appliname1* is not identical to the primary UTM application name.

The statements MUX *partner_name1* ... and CON *partner_name1* are also mutually exclusive.

```
CON  remote_appliname
     [ ,BCAMAPPL=local_appliname ]
     [ ,LISTENER-PORT=number ]
     ,LPAP=lpapname
     ,PRONAM={ processorname | C'processorname'
     }
     [ ,TERMN=termn_id ]


     Unix, Linux and Windows system specific operands

     [ ,T-PROT=RFC1006 ]
     [ ,TSEL-FORMAT={ T | E | A } ]
```

remote_appliname Name of the partner application with which you wish to communicate via the logical connection.

remote_appliname can be up to eight characters in length. Permitted characters are capital letters, numbers and the characters $, # and @. Hyphens are not allowed in names. The first letter must be a capital letter. If lowercase letters are used in a name, you must enter it in single quotes ('...') .

remote_appliname is a mandatory specification.

BS2000 systems:
remote_appliname can be either the BCAM name of a UTM partner application (in the case of a homogeneous link) or the name of a TRANSIT application (in the case of a heterogeneous link). The first letter must be uppercase.

Unix, Linux and Windows systems:
You must specify the T-selector that the partner application uses to sign on to the transport system for *remote_appliname*. The first character must be a letter.

BCAMAPPL= local_appliname

A name for the local application, as defined in the MAX or BCAMAPPL control statement. A BCAMAPPL name may not be specified for which T-PROT=SOCKET is generated.

On Unix, Linux and Windows systems this name must not begin with a ’$’.

The BCAMAPPL name specified in the CLUSTER statement is not permitted here.

Default: If nothing is specified, then the primary application name defined in MAX ..., APPLINAME= is used.

LISTENER-PORT= number

Port number of the partner application.

All port numbers between 1 and 65535 are allowed.

Default: 0 (i.e. no port number)

BS2000 systems:
A port number different from 0 may only be specified if the local application specified in the parameter BCAMAPPL is not generated with T-PROT=NEA.
If 0 is specified, the transport system uses the standard port 102.

Unix, Linux and Windows systems
If OPTION CHECK-RFC1006=YES, then a port number must be entered for LISTENER-PORT.

LPAP= lpapname

Name of the LPAP partner of the partner application with which the connection is to be established. The name of the LPAP partner via which the partner application signs on must be defined using the statement LPAP *lpapname*.

By issuing several CON statements with the same *lpapname*, you can establish parallel connections to the partner application.

Please note, however, that these parallel connections lead to the same partner application (*remote_appliname* and *processorname*).

Mandatory parameter

PRONAM= { processorname | C'processorname' }

Name of the partner host.
The complete name (FQDN) by which the computer is known in the DNS must be specified. The name can be up to 64 characters long.
If the *processorname* contains special characters it must be entered as a character string using C'...'.

Mandatory operand

No distinction is made between uppercase and lowercase notation; KDCDEF always converts the name of the partner computer into uppercase.

TERMN= termn_id

Identifier up to two characters in length, which indicates the type of communication partner. *termn_id* is not queried by openUTM, but is used by the user when querying or grouping terminal types, for example. *termn_id* is entered in the KB header for job-receiving services, i.e. for services started by a partner application in the local application.

Default: A4

T-PROT= This operand is only supported on Unix, Linux and Windows systems.

Address format with which the partner application signs on to the transport system. The following address formats are explained in the "PCMX documentation" (openUTM documentation).

RFC1006 Address format RFC1006

Default: RFC1006

TSEL-FORMAT= This operand is only supported on Unix, Linux and Windows systems.

Format identifier of the T-selector. The format indicator specifies the encoding of the T-selectors in the transport protocol. You will find more information in the "PCMX documentation" (openUTM documentation).

T TRANSDATA format (encoded in EBCDIC)

E EBCDIC character format

A ASCII character format

Default:

T if the character set of the T-selector corresponds to the TRANSDATA format.
E in all other cases

It is recommended to specify a value explicitly for TSEL-FORMAT for operation via RFC1006.

The address of a partner application of a UTM application on Unix, Linux and Windows systems

In order to be able to establish a connection to a partner application, the UTM application must know the address of the partner application. You can enter the address using the following operands:

- *remote_appliname* (address of the partner application in the partner processor)
- PRONAM (real host name or UTM host name of the partner processor)
- LISTENER-PORT (port number for RFC1006)
- T-PROT (the transport protocol used)
- TSEL-FORMAT (format indicator of the T-selector)

See "Providing address information for the CMX transport system (Unix, Linux and Windows systems)".

6.5.11 CREATE-CONTROL-STATEMENTS - create KDCDEF control statements

When regenerating your application, inverse KDCDEF allows you to retain UTM objects in the configuration which were entered dynamically during runtime. Further information can be found in section "Inverse KDCDEF".

The statement CREATE-CONTROL-STATEMENTS generates KDCDEF control statements for the UTM objects which can be entered dynamically, and outputs them to the file *control_statements_file*. During the same KDCDEF run, you can use
control_statements_file as the basis for generation by defining it as an input file using the statement OPTION ..., DATA=*control_statements_file*.

Using the statement CREATE-CONTROL-STATEMENT, you can generate control statements for UTM objects of type TAC, PROGRAM, PTERM, LTERM, USER, CON, LTAC, LSES and KSET.

However, for users generated implicitly, inverse KDCDEF **neither** creates USER statements **nor** adds the user name in the USER= operand in the LTERM statement.

It does not transfer UTM objects to the file *control_statements_file*, which were marked for deletion by administration using KC_DELETE_OBJECT. In a KDCDEF run in which *control_statements_file* is defined as an input file, the names of the deleted UTM objects can be used again.

You can start KDCDEF for inverse KDCDEF with at least one CREATE-CONTROL-STATEMENTS statement and without any further KDCDEF control statements.

> ¡ If you upgrade to a new version, the KDCDEF control statements must first be created in the previous version before being processed in a later version by the KDCDEF generation tool.

CREATE-**C**ONTROL-**S**TATEMENTS	{ *ALL | CON | DEVICE | KSET | LSES | LTAC | PROGRAM | USER } ,FROM-FILE=kdcfile ,TO-FILE=control_statements_file | *LIBRARY-ELEMENT [1] (LIBRARY=<lib-name> ,ELEMENT=<element> [,VERSION=C'<version>' | *HIGH EST-EXISTING | *UPPER-LIMIT | *INCREMENT] [,TYPE=<element-type>])] [,MODE={ CREATE |EXTEND }]

[1] only on BS2000 systems

*ALL	KDCDEF control statements are generated for the following object types: • KSET • LSES • LTAC • TAC • CON • PROGRAM • PTERM • LTERM • USER They cannot be created for other object types.
CON	This creates KDCDEF control statements for the transport connections to remote applications.
DEVICE	KDCDEF control statements are generated for LTERM partners, clients and printers, i.e. for the following object types: • PTERM • LTERM
KSET	KDCDEF control statements are generated for key sets, i.e. for objects of type KSET.
LSES	This creates KDCDEF control statements for the assignment of session names.
LTAC	KDCDEF control statements are generated for transaction codes via which service programs in partner applications are started. These are objects of the type LTAC.
PROGRAM	KDCDEF control statements are generated for programs, service exits transaction codes and TAC queues , i.e. for the following object types: • TAC • PROGRAM
USER	KDCDEF control statements are generated for user IDs, i.e. for objects of type USER. i Please note that passwords cannot be reconstructed. In the case of user IDs with passwords, statements are created with the following format: `USER username, PASS=*RANDOM, ...` In the case of standalone applications, you must use the KDCUPD tool to transfer the passwords to the new KDCFILE after the KDCDEF run has finished. This is also possible for the UTM-F generation variant.

FROM-FILE= kdcfile

Name of the KDCFILE from which the control statements are to be generated.

> The openUTM version of the KDCFILE must match that of the KDCDEF generation tool.

TO-FILE= Specifes where the KDCDEF control statements are to be written to.

control_statements_file The generated KDCDEF control statements are written to the file specified in *control_statements_file*. For *control_statements_file*, you must enter a valid file name. *control_statements_file* can be defined as an input file for the KDCDEF run using the statement OPTION ...,DATA=*control_statements_file*.

*LIBRARY-ELEMENT(...) This parameter is supported on BS2000 systems only.

The KDCDEF control statements are written to the LMS library element specified here. The following restrictions apply:

- Delta elements are not supported.
- UTM always writes records with record type "1".

LIBRARY= <lib-name>

Name of an LMS library. The file name can be up to 54 characters in length. If the library does not yet exist, it is created.
LIBRARY is a mandatory parameter of *LIBRARY-ELEMENT(...).

ELEMENT= <element>

Name of the LMS element.
The element name may be up to 64 characters in length and consists of an alphanumeric string which can be subdivided into multiple substrings separated by periods or hyphens.
ELEMENT is a mandatory parameter of *LIBRARY-ELEMENT(...).

VERSION= Version of the LMS element.

C'<version>' The element version is specified as an alphanumeric string of up to 24 characters in length which can be subdivided into multiple substrings separated by periods or hyphens.

*HIGHEST-EXISTING The statements are written to the highest version of the specified element present in the library.

*UPPER-LIMIT The statements are written to the highest possible version of the specified element. LMS indicates this version by means of an "@".

*INCREMENT

A new version is created for the specified element. *INCREMENT may only be specified if MODE=CREATE.

Default value:

- *HIGHEST-EXISTING in MODE=EXTEND
- *INCREMENT in MODE=CREATE

> If MODE=CREATE and VERSION is not equal to *INCREMENT then any existing element is overwritten with the specified version.

TYPE=

<element-type>

Type of LMS element. An alphanumeric string of up to 8 characters in length can be specified for the type.

Default value: S

> KDCDEF does not check whether the specifications for ELEMENT, VERSION or TYPE comply with the LMS syntax rules. For further information on the syntax rules for the names of LMS elements and a specification of version and type, see the manual "LMS SDF Format".

MODE=

Write mode of the file containing the generated KDCDEF control statements

CREATE

The file specified in *control_statements_file* is created.

On BS2000 systems the file is created as a SAM file or an LMS library element. The following applies:

- If a file with the same name already exists, this must be a SAM file. This SAM file is then overwritten.
- If an element of the same name already exists and if *HIGHEST- EXISTING or *UPPER-LIMIT is specified for VERSION=C‘<version>‘ then the element is overwritten.

If a file with the same name already exists on Unix, Linux or Windows systems, it is overwritten.

EXTEND

The generated control statements are appended to the existing *control_statements_file*. If this file does not exist, it is created.

If an LMS library is specified in the BS2000 system then the library must already exist. In this case, an existing element of the specified version is extended. If the element does not yet exist in this version then it is created.

6.5.12 DATABASE - define a database system (BS2000 systems)

The DATABASE control statement allows you to define the database systems with which the UTM application is to coordinate.

Each database system must be defined in a separate DATABASE statement. By issuing several DATABASE statements for the same database system, you can assign several entry names to that database system.

The DATABASE statement can be issued several times. It is thus possible to define up to three (in a special release, up to eight) different database systems.

DATABASE	[ENTRY=entryname] [,USERID=username | C‘username‘] [,PASSWORD=C'password'] [,LIB=omlname | LOGICAL-ID(logical-id) }] [,TYPE={ UDS | SESAM | LEASY | DB | XA | CIS }] [,XA-INST[-NAME]=inst-name]

ENTRY= entryname

Entry name of the database. The following default values apply:

$UNIBASE	if TYPE = UDS
SESAM	if TYPE = SESAM
CIS	if TYPE = CIS
LEASY	if TYPE = LEASY
DB	if TYPE = DB

When generating the XA connection with TYPE=XA in openUTM on BS2000 systems, the name of the XA switch as it is provided by the database system **must** be specified with the ENTRY parameter. It is possible to generate several XY switches in the DATABASE statement.

> ¡ A database connection to Oracle must be generated with TYPE = XA.

Other entry names (e.g. SQLUDS for UDS/SQL) can be found in the manuals for the respective database systems.

USERID= username | C‘username‘

Specifies a user name for the database system. The user name can be up to 30 characters in length.

This functionality is only provided for Oracle databases. openUTM passes this name to the database system in the Open string.

If a user name is to be passed to the database system in lowercase characters, then you must use the format C'username'.

> Alternatively, the user name can be transferred to the database system by means of start parameters.
>
> For XA databases (TYPE=XA), it is possible to modify the user name and/or the password by means of dynamic administration.

PASSWORD= C'password'

Specifies a password for the database system. The password can be up to 30 characters in length.

This functionality is only provided for Oracle databases. openUTM passes the password to the database system in the Open string.

Alternatively, the password can be transferred to the database system by means of start parameters.

LIB= Specifies library from which the connection module for the database system is dynamically loaded.

omlname OML name which the connection module for the database system is to be loaded dynamically. *omlname* can be up to 54 characters in length.

LOGICAL-ID(logical-id)

Specifies that a search is to be made for the connection module in the IMON installation path for the database system and that the module is to be loaded from there.
logical-id is a name up to 15 characters long. It may be specified only for SESAM/SQL and UDS/SQL; it is SYSLNK for both database systems, refer also to the notes in section "*Notes on using LOGICAL-ID"*.

If you do not specify LIB= , then LIB= is set to TASKLIB. This does not correspond to the SET-TASKLIB command, rather a library named TASKLIB must exist. Dynamic loading of the connection module from the library assigned with SYSFILE-TASKLIB is not supported.

> During the dynamic load, the DBL searches for the connection module first in the library that you specified in LIB= . If this library does not exist, then the DBL aborts the search. If the library exists but the connection module is not found there, then the DBL searches the alternative libraries. These libraries are the libraries that have been assigned a file chain name BLSLIB*nn* (0<=*nn*<=99).
>
> If several DATABASE statements are issued with the same TYPE in order to generate a number of entries for the same database, the connection module is loaded from the library specified in the LIB operand of the first DATABASE statement with the relevant TYPE.

Notes on using LOGICAL-ID

- LIB=LOGICAL-ID(*logical-id*) may be specified only if the database system was correctly installed with IMON. If the database system was not installed with IMON, you must either statically link the connection module (without the LIB= operand) or you must specify LIB= *omlname*.
- Using LOGICAL-ID(*logical-id*) instead of *omlname* has the advantage that the openUTM application is then independent of the installation paths and library names of the database system.
- If you specify LIB=LOGICAL-ID(SYSLNK) and if several product versions are installed, the most recent version is used by default.
- If you do not want the most recent version to be loaded, you must either specify the library of a less recent version using LIB=*omlname* or you must assign the version before starting the openUTM application (using the SELECT-PRODUCT-VERSION command of IMON).
- If an error occurs when searching for the connection module in the IMON installation path, application start is aborted and the error is logged to SYSOUT.

TYPE= This identifies the database system.

With TYPE=DB you can also connect to database systems other than those named above. This is only possible when the database system supports the IUTMDB interface.

Default: UDS

XA-INST-NAME= This parameter is permitted only if TYPE=XA was specified.

inst-name is the local name for the XA instance which is 1 to 4 characters long.

If more than one DATABASE statement with TYPE=XA is generated for an application, the statements must differ from XA-INST-NAME in their values. In the case of applications with just one XA database, the parameter can be omitted.

The string specified for XA-INST-NAME must be specified after the string ".RMXA" in the prefix for the start parameters for this database.

Example:

Specification in the DATABASE statement:

```
DATABASE ....,TYPE=XA,XA-INST-NAME=DB1
```

Specification in the start parameters for this data base:

```
.RMXADB1 RMXA RM="rm-name",OS="open-string",...
```

Default: blanks

6.5.13 DEFAULT - define default values (BS2000 systems)

The DEFAULT control statement allows you to define default values for the operands of a KDCDEF control statement. A default operand value set using DEFAULT applies until the next DEFAULT statement is issued for the same operand in the same control statement. If you subsequently wish to reset the default value to the UTM standard setting, you must reassign this standard setting using the DEFAULT statement. If this is not possible (e.g. FORMAT = blanks), then the default value is set in the (STD) or *STD entry.

Statement-specific default values offer the following advantage:
If you issue a control statement several times (e.g. PTERM), there is no need to specify the same operand values over and over again in each statement (e.g. the processor name in PRONAM).

> When porting BS2000 systems openUTM applications to Unix, Linux or Windows systems, please note that the DEFAULT statement is not supported by openUTM on Unix, Linux and Windows systems.

`DEFAULT`	`control-statement operand [ ,operand ] [ ,... ]`

control-statement

KDCDEF control statement for which new default values are to be defined in this DEFAULT statement. The following operands are dependent on this control statement, and apply only for this control statement class. Please note that the PROGRAM and AREA statements form a **single** class, i.e. modified default values of the PROGRAM statement also apply for the other statements in this class.

You must insert at least one blank between *control_statement* and the following operands. The table on the next page shows the control statements that can be specified here.

operand ,...

One or more operands of the KDCDEF control statement *control_statement*. Each operand is separated by a comma. The table on the next page shows the operands permitted for *control_statement*.

Permitted control statements	Permitted operands
CON	BCAMAPPL={*local_applname*|(STD)} LPAP={*lpapname*|(STD)} PRONAM={*processorname* | C'*processorname*'} TERMN=*termn_id*
LPAP	DEAD-LETTER-Q={NO | YES} NETPRIO=*netprio* QLEV=*queue_level* STATUS={ON|OFF} SESCHA=*sescha_name*
LSES	LPAP=*sessionname* NODE-NAME=*node_name*
LTAC	LPAP=*lpapname* LTACUNIT=*ltacunit* STATUS={ON|OFF} TYPE={D|A} WAITTIME=(*time1*,*time2*)
LTERM	ANNOAMSG={Y|N} FORMAT={*formatname*|(STD)} KERBEROS-DIALOG={YES | NO} LOCALE={ ([*lang_id*], [*terr_id*],[*ccsname*]) | *STD} NETPRIO=*netprio* PLEV=*print_level_number* QAMSG={Y|N|(STD)} QLEV=*queue_level_number* RESTART={YES|NO} STATUS={ON|OFF} USAGE={D|O}
LOAD-MODULE	LIB=*libname* LOAD-MODE=*loadmode* VERSION={*version* | *HIGHEST-EXISTING | *UPPER-LIMIT}
OSI-CON	ACTIVE={YES|NO} LOCAL-ACCESS-POINT=*access-point_name*
OSI-LPAP	APPLICATION-CONTEXT=*application_context* DEAD-LETTER-Q={NO | YES} IDLETIME=*time* QLEV=*queue_level_number* STATUS={ON|OFF} TERMN=*termn_id*

Permitted control statements	Permitted operands
PROGRAM	COMP=*compiler* LOAD-MODULE={*lmodname*|*STD}
PTERM	BCAMAPPL=*local_appliname* CONNECT={YES | NO} ENCRYPTION-LEVEL={NONE | 3 | 4 | TRUSTED} IDLETIME=*time* MAP={USER | SYSTEM | SYS | SYS1 | SYS2 | SYS3 | SYS4} PRONAM={*processorname* | C'*processorname*' | *RSO} PROTOCOL={N|STATION} PTYPE={*partnertyp* | *RSO | *ANY} STATUS={ON|OFF} TERMN={*termn_id*|(STD)} USAGE={D|O} USP-HDR={ALL | MSG | NO}
SESCHA	CONNECT={Y|N} CONTWIN={Y|N|(STD)} DPN={*instance_name*|(STD)} IDLETIME=*time* PLU={Y|N|(STD)} PACCNT=*number*
TAC	ADMIN={Y | N} CALL={BOTH | FIRST | NEXT | (STD)} DEAD-LETTER-Q={NO | YES} ENCRYPTION-LEVEL={NONE | 2 } EXIT={*exit* | (STD) } PGWT={NO | YES} PROGRAM={*program_name*|(STD)} QLEV=*queue_level_number* QMODE = {STD | WRAP-AROUND} RUNPRIO=*priority* SATADM={NO|YES} SATSEL={BOTH|SUCC|FAIL|NONE} STATUS={ON | OFF | HALT | KEEP} TACCLASS={*class*|(STD)} TACUNIT=*tacunit* TCBENTRY={*name_of_tcbentry-statement* | (STD)} TIME={*time1* | (*time1*,*time2*)} TYPE={D | A | Q}

Permitted control statements	Permitted operands
TPOOL	ANNOAMSG={ Y | N } BCAMAPPL=*appliname* ENCRYPTION-LEVEL={NONE | 3 | 4 | TRUSTED} FORMAT={*formatname*|(STD)} IDLETIME=*time* KERBEROS-DIALOG={YES | NO} LOCALE={ ([*lang_id*], [*terr_id*],[*ccsname*]) | *STD } MAP={ USER | SYSTEM | SYS | SYS1 | SYS2 | SYS3 | SYS4} NETPRIO={ MEDIUM | LOW } NUMBER=*number1* PRONAM={ *processorname* | C'*processorname*' | *ANY } PROTOCOL={ N | STATION } PTYPE={ *partnertyp* | *ANY } QLEV=*queue_level_number* TERMN={ *termn_id* | (STD) } USP-HDR={ALL | MSG | NO}
USER	FORMAT={*formatname*|(STD)} LOCALE={ ([*lang_id*], [*terr_id*],[*ccsname*]) | *STD } PERMIT={NONE | ADMIN | SATADM | (ADMIN,SATADM)} PROTECT-PW=(*length,level_of_complexity*,*max_time,min_time*) QLEV=*queue_level_number* QMODE = {STD | WRAP-AROUND} Q-READ-ACL = *keysetname* Q-WRITE-ACL = *keysetname* RESTART={YES | NO} SATSEL={BOTH | SUCC | FAIL | NONE} STATUS={ON | OFF}

6.5.14 EDIT - define edit options (BS2000 systems)

The EDIT control statement allows you to combine screen functions and screen output properties in line mode (edit options) in groups known as edit profiles. It also enables you to assign names to these edit profiles, which can then be used to address a set of edit options from a program unit.

The EDIT statement can be issued several times within a generation run. However, a different name (*name* operand) must be specified in each EDIT statement.

The edit profile names are specified in the KCMF field of the MPUT, MGET, DPUT, FPUT and FGET calls at the programming interface, where a blank is entered as the format control character.

openUTM interprets the entries in the KCMF field as follows:

No edit profiles generated	Edit profiles generated
If a blank is entered as the format control character, openUTM ignores the remaining characters in the field.	If a blank is entered as the format control character, the remaining characters in the field must contain either the name of a valid edit profile or further blanks.

A detailed description of the operands described below can be found in the „TIAM“ User Guide. Further information on working with edit profiles can be found in the openUTM manual „Programming Applications with KDCS”.

```
EDIT  name
      [ ,BELL={ NO | YES } ]
      [ ,CCSNAME=ccsname ]
      [ ,HCOPY={ NO | YES } ]
      [ ,HOM={ NO | YES } ]
      [ ,IHDR={ NO | YES } ]
      [ ,LOCIN={ NO | YES } ]
      [ ,LOW={ YES | NO} ]
        ,MODE={ EXTEND | INFO | LINE | PHYS | TRANS }
      [ ,NOLOG={ NO | YES } ]
      [ ,OHDR={ NO | YES } ]
      [ ,SAML={ NO | YES } ]
      [ ,SPECIN={ C | I | N } ]
```

name — Alphanumeric name up to seven characters in length for the set of edit options to be defined.

BELL= — This specifies whether or not an acoustic alarm is triggered on the terminal when a message is output.

CCSNAME= ccsname

(**c**oded **c**haracter **s**et **name**)
Name of the character set (CCS name) used to format a message. This name can be up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). The character set must be compatible with an ISO character set supported by the terminal to which the message is directed. During generation, KDCDEF cannot check the validity of the CCS name under the BS2000 system or the compatibility condition.

A CCS name must not be assigned to the edit profile if the value TRANS (transparent mode) is defined for the MODE operand.

> ¡ If the edit profile is used to output messages to an RSO printer, only the CCSNAME= parameter of the edit profile is evaluated.

HCOPY= (**h**ard **copy**)
This specifies whether the output message is to be logged on a connected hardcopy printer in addition to being displayed on the terminal.

HOM= (**hom**ogeneous)
This specifies whether the output message is to be output unstructured, i.e. in homogeneous format. If you enter NO here, the message is output in a structured format, i.e. in non-homogeneous format. In this case, a logical line is regarded as an output unit.

IHDR= (**i**nput **h**eade**r**)
This specifies whether the header of the input message is to be transferred to the program unit.

LOCIN= (**loc**al parameter **in**put)
This operand applies only for terminals that support local parameters (e.g. 9763). If you enter YES here, local attributes in the input message are forwarded to the user as logical control characters. If you enter NO here, local attributes are removed from the input message and are not forwarded. LOCIN=YES is permitted only if MODE=EXTEND.

LOW= (**low**ercase)
This specifies whether lowercase letters are permitted in the input message transferred to the program unit. If you enter NO here, the system converts all lowercase letters into uppercase.

MODE=

EXTEND (**extend**ed line mode)
This specifies whether the message is to be output in extended line mode.
If you enter MODE=EXTEND, the value YES is permitted only for the BELL, LOW, and LOCIN edit options. The value N must be entered for the SPECIN operand.

INFO The message is to be output in a special information line (system line) without overwriting important data at the terminal.

This specification is primarily intended for application programs that send "asynchronous" messages to terminals without knowing what is currently being displayed at the terminal. At terminals with a hardware display line (e.g. DSS 9749, 9750, 9763), the data is always output protected in a hardware system line; in all other cases, it is output in the same way as a normal line mode message.

LINE (**line** mode)
The message is to be output in line mode. It can be structured using logical control characters, and is formatted by the system. If you enter MODE=LINE, the value NO must be entered for IHDR, OHDR and LOCIN.

PHYS (**phys**ical mode)
The message is to be output or read in physical mode, i.e. without being formatted by the system. If you enter MODE=PHYS, the value YES is permitted only for the IDHR, LOW and OHDR edit options. The value N must be entered for the SPECIN operand.

This specification should not be used for messages output on a printer. Physical messages to a printer can only be implemented using a format exit.

TRANS (**trans**parent mode)
The output message is to be transferred in transparent mode. If you enter MODE=TRANS, the value YES must not be specified for any other edit option.

The value N must be entered for the SPECIN= operand.
The CCSNAME= operand must not be specified.

NOLOG= (**no log**ical characters)
This specifies how the system is to handle non-printable characters.

YES The logical control characters are not evaluated. All characters less than X’40’ in EBCDIC code are replaced by alternate characters (SUB). Only printable characters are allowed through.

NO All logical control characters are evaluated. Special physical control characters are allowed through. All other characters less than X’40’ are replaced by alternate characters (SUB). Printable characters are allowed through.

Default: NO

OHDR= (**o**utput **h**ea**d**e**r**)
This specifies whether the output message contains a header. The length of the message header + 1 must be entered in binary format in the first byte of the message.

SAML= (**sam**e **l**ine)
This applies only for printer stations.
If SAML=YES, the message is not preceded by a line feed. If SAML=NO applies, the message begins at the start of the next line.

SPECIN= (**spec**ial **in**put)

C (**c**onfidential)
This specifies whether the display of input data is to be suppressed on the terminal, thus protecting confidentiality.

I (**i**d-card)
This specifies whether input data is to be entered via the ID card reader.

N (**n**ormal)
The terminal requires normal input.

6.5.15 EJECT - initiate a page feed in the log

The EJECT control statement allows you to initiate a page feed in the log. The EJECT line itself is not logged or counted.

EJECT	

6.5.16 END - terminate KDCDEF input

The END control statement identifies the end of the sequence of control statements, and is the last statement entered.

END	

> If a file with OPTION DATA=*filename* is defined as a KDCDEF input file and contains an END statement, KDCDEF input is terminated as soon as this statement is processed.

6.5.17 EXIT - define event exits

The EXIT control statement allows you to define event exits which are used in the application, except for the event exits VORGANG and HTTP.

For the event exits FORMAT and INPUT, you may only specify a single EXIT statement for each KDCDEF run. For the event exits START and SHUT, you may specify up to eight EXIT statements. However, the specifications for the PROGRAM= operand must differ for the EXIT statements.

When starting or terminating a UTM process, all of the programs defined as START or SHUT exits are called one after the other. The sequence of the EXIT statements in the KDCDEF run determines the sequence in which openUTM activates the START or SHUT exit program.

Further information on event exits can be found in the openUTM manual „Programming Applications with KDCS”.

Event exits on BS2000 systems:
The event exits START, SHUT, INPUT and FORMAT must not be assigned to a load module generated with LOAD-MODULE LOAD-MODE=ONCALL.

> The event services MSGTAC, BADTACS and SIGNON must be defined using the TAC statement.

```
EXIT  PROGRAM=objectname
      ,USAGE={ START |
               SHUT |
               ( INPUT, { ALL | FORMMODE | LINEMODE | USERFORM[1] } ) |
               FORMAT[1] }
```

[1]FORMAT and USERFORM are only permitted on BS2000 systems

PROGRAM= name

Name of the program containing the functions to be executed for the event exit. A PROGRAM statement with this name (*objectname*) must be issued.

USAGE= Type of event exit

START Used as event exit START

SHUT Used as event exit SHUT

INPUT Used as event exit INPUT
Additionally you must specify the type of INPUT exit:

ALL Event exit INPUT, which handles messages of all format control characters as well as LINEMODE messages.

> If you specify ALL here, this is the only event exit INPUT of the application. Further INPUT event exits cannot be defined.

FORMMODE Event exit INPUT for +, *, and #formats

LINEMODE	Event exit INPUT for LINEMODE messages
USERFORM	Event exit INPUT for -formats
FORMAT	Used as event exit FORMAT

6.5.18 FORMSYS - define the format handling system (BS2000 systems)

The FORMSYS control statement allows you to define the format handling system. Only the first FORMSYS statement is evaluated.

FORMSYS	[TYPE=typ] [,ENTRY=entryname] [,LIB=omlname]

TYPE= typ

This identifies the format handling system.
Only the value FHS is supported.

Default: FHS

ENTRY= entryname

Entry name for the format handling system

Default value: KDCFHS with TYPE=FHS

LIB= omlname

Designates the object module library (OML) from which the connection module for the format handling system is loaded. *omlname* can be up to 54 characters in length.

If you do not specify LIB= , then LIB= is set to TASKLIB. This does not correspond to the SET-TASKLIB command, rather a library named TASKLIB must exist. Dynamic loading of the connection module from the library assigned with SYSFILE-TASKLIB is not supported.

> **i** During the dynamic load, the DBL searches for the connection module first in the library that you specified in LIB= . If this library does not exist, then the DBL aborts the search. The DBL does not abort the search if LIB was not specified but was preset to TASKLIB and no file with this name exists. If the library exists but the connection module is not found there, then the DBL searches the alternative libraries. These libraries are the libraries that have been assigned a file chain name BLSLIB*nn* (0<=*nn*<=99).

6.5.19 HTTP-DESCRIPTOR - define a HTTP Descriptor

With the HTTP-DESCRIPTOR statement, a mapping of the path received in an HTTP request to a TAC is defined and additional processing parameters can be specified. The specifications in the HTTP-DESCRIPTOR statement are used by UTM after receiving an HTTP request to determine the TAC to which the message is to be sent and to control the processing of the message.

If no suitable HTTP-DESCRIPTOR statement is defined for the path of an HTTP request, UTM performs a standard conversion of the messages for this request if the path can be mapped directly to a TAC defined for the application.

HTTP- DESCRIPTOR	http-descriptor-name [,BCAMAPPL = bcamappl | *ALL] [,HTTP-EXIT = program-name | *NONE | *SYSTEM] [,PATH = C'path' | C'/*'] ,TAC = tac [,USER-AUTH = *NONE | *BASIC] ***Only on BS2000 systems*** [,CONVERT-TEXT = *YES | *NO]

http-descriptor-name
: The parameter *http-descriptor-name* has only local application meaning. It assigns a local name to the HTTP-DESCRIPTOR. This is required, for example, at the administration interface. The name can be a maximum of 8 characters long.

BCAMAPPL = bcamappl

: With BCAMAPPL, you can specify the name of a BCAMAPPL statement.

  If BCAMAPPL is specified, the HTTP-DESCRIPTOR statement is only valid for HTTP connections via this BCAMAPPL.

  If *ALL is specified, the HTTP-DESCRIPTOR statement applies to all HTTP connections.

  The BCAMAPPL statement can also be used to specify the schema *(HTTP/HTTPS)* for which this HTTP-DESCRIPTOR is to be valid.

  The name can be a maximum of 8 characters long.

  Default: *ALL

CONVERT-TEXT =
: The parameter CONVERT-TEXT may only be specified in BS2000 systems.

  The CONVERT-TEXT parameter specifies whether UTM is to perform a code conversion for text messages or not. UTM evaluates the specifications in the Content-Type header of an HTTP request and and the mapping to a code conversion tables defined with the CHAR-SET statement. A code conversion of an HTTP message is only performed, for example, if a code conversion table is assigned to the character set specified in the Content-Type header using the CHAR-SET statement.

  *YES
  : UTM should perform a code conversion.

  *NO
  : UTM should not perform a code conversion.
  *NO is the default value.

HTTP-EXIT =

The HTTP-EXIT parameter can be used to define a user program that is to be called by UTM to reformat the input and output messages.
The program specified in HTTP-EXIT must be defined with a PROGRAM statement.

The name of the user program must not exceed 32 characters.

*NONE

If *NONE is specified for HTTP-EXIT, web-aware programs that can process messages from HTTP clients directly, that is, without reformatting, are identified.
*NONE is the default value.

*SYSTEM

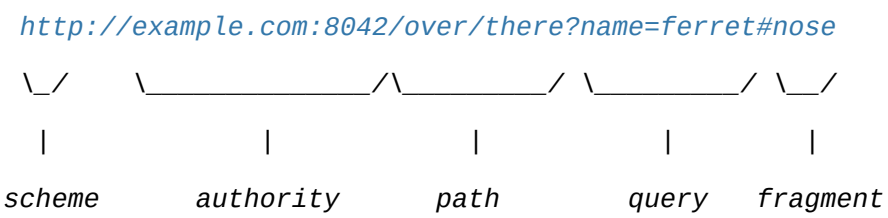
With *SYSTEM you can specify that UTM is to convert output messages into HTML format.

TAC = tac

The parameter TAC determines the TAC and therefore the part-program that is to be called for requests with the path specified in this statement. The TAC must be defined with a TAC statement. Only one dialog TAC may be specified. The same TAC may be specified in different HTTP-DESCRIPTOR statements. The TAC can be a maximum of 8 characters long.

PATH =

According to RFC 3986 "Uniform Resource Identifier" a URI is structured as follows:



The path of an HTTP request is used to address a resource. UTM uses the path to determine the part-program to which an HTTP request is to be sent.

If no TAC can be determined in any way for an incoming HTTP request, for example, because no standard path (C'/*') has been generated, the HTTP request is rejected with status code *404 Not Found*.

A "*" as the last character in the parameter PATH has the meaning of a wildcard character, i.e. the prefix of the path is defined via such a declaration. If the beginning of the path of an HTTP request matches a path prefix defined in this way, then this HTTP-DESCRIPTOR is evaluated for the request.

C'path'

The parameter *path* must fulfill the following conditions:

- First character must be a "/".
- The path must not contain the character string "//".
- The path must be at least two characters long.
- The characters ":", "?" and "#" must not appear in the path.
- A '*' must not be more than the last character in the path.
- If the last character in the path is a '/', it is ignored.
- The maximum length of the path is 254 characters.
- The specification of the character string in the parameter PATH must be unique for each BCAMAPPL.

KDCDEF stores the path specified here in normalized form. This means that %-coded substitute representations of unreserved characters and spaces are converted to their equivalent one-character representation. For % codes and Unreserved Characters see RFC 3986.

C'/*'

An HTTP-DESCRIPTOR statement with PATH=C'/*' defines the standard behavior for one or all BCAMAPPLs. Such a declaration is used by UTM if a TAC cannot be determined for an HTTP request in any other way.
C'/*' is the default value.

USER-AUTH =

The USER-AUTH parameter specifies which authorization mechanism clients must use for this application.

The value set here for this parameter applies to all HTTP messages to which a TAC is assigned using this HTTP-DESCRIPTOR statement.

If the UTM application is generated without users, only the value *NONE may be specified for USER-AUTH.

*BASIC

The Basic Authentication Scheme from RFC 2617 is to be used to transfer authentication data. UserId and password are separated by a colon and Base64 encoded in the authorization header of an HTTP request.

If Basic Authorization is defined for an HTTP request, but no Authorization Header is contained in the HTTP request, UTM requests authentication data using a response with status code *401 Unauthorized*.

*NONE

If *NONE is specified, the client does not have to pass any authentication data. UTM uses the connection user for such a request if the client does not send authentication information in the HTTP request itself.

Default: *NONE

6.5.20 KSET - define a key set

The KSET control statement allows you to combine the key codes of an application, which were defined for data access control, to form a logical key set. You can specify several control statements for a single key set.

KDCDEF implicitly generates the KDCAPLKS key set, which by default contains all key codes.

KSET	keysetname ,KEYS={ (key1,key2,... key *n*) | MASTER }

keysetname Name of the defined key set up to eight characters in length.

You can assign this key set

- to a user (in a USER statement in section "USER - define a user ID")
- to an LTERM partner (in a LTERM statement in section "LTERM - define an LTERM partner for a client or printer")
- to a partner application (in an LPAP or OSI-LPAP statement in section "LPAP - define an LPAP partner for distributed processing based on LU6.1" or "OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP")
- to a TAC (in the TAC statement in section "TAC - define the properties of transaction codes and TAC queues")
- to an LTAC (in the LTAC statement in section "LTAC - define a transaction code for the partner application")
- to a TPOOL (in the TPOOL statement in section "TPOOL - define an LTERM pool" or "Format and uniqueness of object names")

After the connection has been established, the key set of the LTERM or (OSI-)LPAP partner assigned to the connection is available to the client or partner application. After signing on to the application, the key set of the user ID is available to the client or partner application.
The lock/key code and the access list concept are described in detail in the openUTM manual “Concepts und Functions”. An introduction to access control can be found in section "Lock/key code concept".

KEYS = (key1,..., key*n*)

Key or access codes of the key set *keysetname*

List of numbers between 1 and the maximum value permitted by the application (MAX ..., KEYVALUE=*number*). These numbers correspond to the key codes contained in this key set.

A key or access code grants access to a resource secured with a lock code or an access list, provided the key code and lock code match or the access code is contained in the access list.

You can specify up to 60 key codes/access codes in each KSET statement. If a key set contains more than 60 key codes, you must issue another KSET statement with the same *keysetname*.

If you only specify one key code, you can omit the parentheses.

If you enter the value 0 for *key*, this is ignored by openUTM. No message is output.

MASTER The MASTER key set contains all the key codes/access codes of the application.

6.5.21 LOAD-MODULE - define a load module (BLS, BS2000 systems)

The LOAD-MODULE control statement allows you to define the name, version and properties of load modules. If you use the BLS interface, this statement must be issued for all load modules that can be exchanged or loaded as independent units during the program run. Each load module must be defined in a separate LOAD-MODULE statement.

The load modules that can be processed with BLS are either LLMs (link and load modules) or OMs (object modules). However, it is recommended that the program components and data areas to be loaded dynamically are linked to LLMs (see the BLS manuals).

A load module can contain several program units and data areas, which are defined using PROGRAM or AREA statements. You can assign one or more PROGRAM and/or AREA statements to a single LOAD-MODULE statement. This takes place on the basis of the load module name *lmodname*, which must also be entered in the LOAD-MODULE operand of the PROGRAM or AREA statement. However, it is also possible to generate LOAD-MODULE statements without assigning a PROGRAM or AREA statement (e.g. load modules that contain parts of the runtime system of a programming language).

At least one LOAD-MODULE statement must be generated if the "program exchange" function (KDCAPPL PROGRAM=NEW) is to be used on BS2000 systems.

When starting the UTM application, the load modules are loaded in accordance with the sequence of LOAD-MODULE statements and the value of the LOAD-MODE operand. The load sequence is as follows:

- The basic part of the application, including all load modules linked in statically to the application program (LOAD-MODE=STATIC).
- All load modules loaded into a global common memory pool when starting the UTM application. These are generated with LOAD-MODULE LOAD-MODE=(POOL, *poolname,...*) and MPOOL *poolname*,SCOPE=GLOBAL. The common memory pools are loaded in accordance with the sequence of MPOOL statements in the generation run. Within a pool, the sequence of LOAD-MODULE statements that refer to this pool applies.
- All load modules loaded into a local common memory pool when starting the UTM application. These are generated with LOAD-MODULE LOAD-MODE=(POOL, *poolname,...*) and MPOOL *poolname*,SCOPE=GROUP. The pools are loaded in accordance with the sequence of MPOOL statements. Within a pool, the sequence of LOAD-MODULE statements that refer to this pool applies.
- All load modules to be loaded dynamically as independent units during startup. These are generated with LOAD-MODULE LOAD-MODE=STARTUP. The load modules are loaded in accordance with the sequence of LOAD-MODULE statements defined in this way.

Load modules generated with LOAD-MODE=ONCALL are loaded the first time an assigned program unit is called.

Please note the following:

- Load modules containing TCB entries **cannot** be exchanged.
- When dynamically linking a load module with the generation ALTERNATE-LIBRARIES=YES, you must ensure that only RTS modules are actually linked. This is because when load modules are exchanged, only the load module itself is removed from memory. If the load module is used to dynamically load other modules using the autolink function, these modules remain in memory following the exchange process even though shared data structures, for example, have been modified.
- When linking with the SYSLNK.CRTE.PARTIAL-BIND library, the entry ALTERNATE-LIBRARIES=YES is not required for load modules that only contain C code and possibly data objects (areas), and should therefore be avoided.

LOAD-MODULE	lmodname [,**ALT**ERNATE-**LIB**RARIES={ NO | YES } [,LIB=libname] [,LOAD-MODE={ STARTUP | ONCALL | STATIC | (POOL,poolname,{ NO-PRIVATE-SLICE | STARTUP | ONCALL }) }] ,VERSION={ version | ***HIGH**EST-**EXIST**ING | ***UP**PER-**LIM**IT }

lmodname

Name of the load module up to 32 characters in length

This name is subject to the same rules as the element names of a program library (see also "Format of names").

ALTERNATE-**LIB**RARIES=

This is used to control the autolink function when dynamically linking the private slice of the load module.

NO

The autolink function is not executed when linking the load module.

Default: NO

YES

The BLS autolink function is activated.
For instance, if a load module requires other RTS modules for exchange purposes and these have not yet been loaded into memory, this function is used to load these RTS modules dynamically. Before starting the UTM
application, the required RTS libraries must be reserved with Linkname BLSLIB*nn* (00 <= *nn* <= 99). If open external references cannot be resolved by the loaded modules, these libraries are then searched in ascending order
(as specified in *nn*) for appropriate definitions when dynamically loading the load modules.

ALTERNATE-LIBRARIES=YES may only be used to dynamically load RTS modules and not to dynamically load user programs.

Further information on the Autolink function can be found in the openUTM manual “Using UTM Applications on BS2000 Systems”.

> **i** The operand values LOAD-MODE=STATIC / (POOL,*poolname*,NO-PRIVATE-SLICE) cannot be combined with ALTERNATE-LIBRARIES=YES. Such a combination will be rejected by the KDCDEF run.

LIB= libname

Program library from which the load module is to be loaded dynamically. *libname* can be up to 54 characters in length.

If LOAD-MODE = STATIC, the LIB= operand is ignored. In all other cases, you must assign a value to LIB= either in the LOAD-MODULE statement or in a preceding DEFAULT statement.

LOAD-MODE= Load mode of the load module

STARTUP The load module is loaded dynamically as an independent unit when the application is started. External references from the subsystem, from class 3/4 memory, and from all other modules of the UTM application which are already loaded are resolved. For runtime system functions, see also the description of the operand ALTERNATE-LIBRARIES=YES.

> **i** Load modules which are generated with LOAD-MODE=STARTUP and which contain TCB entries must not be exchanged during runtime.
>
> Default: STARTUP

ONCALL The load module is loaded dynamically as an independent unit the first time a program unit or conversation exit assigned to the load module is called. External references from class 3/4 memory and from all other modules of the UTM application which are already loaded are resolved.

> **i** Load modules containing TCB entries must not be generated with LOAD-MODE=ONCALL.
>
> If you are working with several processes, this load module must not be overwritten in the library LIB=*libname* during runtime. Otherwise, different versions of the load module will be executed during the application run.

STATIC The load module must be linked in statically to the application program.

(POOL,poolname, NO-PRIVATE-SLICE)

The memory pool is defined using the MPOOL statement.

poolname can be up to 50 characters in length.

When the application is started, the load module is loaded into the common memory pool *poolname*. It is not divided into public and private slices. A private slice is therefore not linked (even statically) into the application program.

(POOL,poolname, STARTUP)

When the application is started, the public slice of the load module is loaded into the common memory pool *poolname*. The private slice belonging to the load module is then loaded into the local task memory.

(POOL,poolname, ONCALL)

When the application is started, the public slice of the load module is loaded into the common memory pool *poolname*. The private slice belonging to the load module is then loaded into the local task memory when the first program unit assigned to this load module is called.

Only external references from class 3/4 memory, from the subsystems, and from the local memory pool are resolved.

VERSION=version | *HIGHEST-EXISTING | *UPPER-LIMIT

Version of the load module. *version* can be up to 24 characters in length.

*UPPER-LIMIT

If VERSION=*UPPER-LIMIT (or VERSION=@) is specified then the BLS addresses the load module *lmodname* in a PLAM library, which was last entered in this PLAM library without an explicit version specification. If you work with explicit versions in LMS, you **cannot** use @ as the load module version.

The rules governing the versions of elements in a program library also apply to name allocation. However, there is one limitation: if *version* contains the character “.” then the version must start with a letter.

*HIGHEST-EXISTING

Every time the application is started and at every application exchange, the highest version of this module that exists in the library is loaded, i.e.

- For the first task (application start) or the initiating task (before application exchange), UTM determines the highest current element version for all load modules generated with VERSION=*HIGHEST-EXISTING.
- This element version is then also used to load modules that are not loaded until later, e.g. because they are generated with LOAD-MODE=ONCALL.
- When starting follow-up tasks for an application and when reloading the application program after a PEND ER, the version of this module that is loaded is the one that is already loaded by the other tasks in the application or that was loaded in this task before the PEND ER.

For LOAD-MODULE statements generated with LOAD-MODE=STATIC, specifying VERSION=*HIGHEST-EXISTING is **not** permitted.

6.5.22 LPAP - define an LPAP partner for distributed processing based on LU6.1

The LPAP control statement allows you to define a logical access point for the partner application in the local application. An LPAP statement is only required if communication with the partner application is to be carried out using the LU6.1 protocol. This logical access point is known as an LPAP partner. For each LPAP partner, you must define a logical name, possibly administration authorization for the partner application, maximum values for the message queue of the LPAP partner, and logical properties for communication with the partner application based on the LU6.1 protocol.

For information about generating LU6.1 connections see "Distributed processing via the LU6.1 protocol".

The CON statement is used to assign a real partner application to the LPAP partner (see the CON statement in the section "CON - define a connection for distributed processing based on LU6.1"

LPAP	lpapname [,BUNDLE = master_lpap_name] [, DEAD-LETTER-Q={ NO | YES } [,KSET=keysetname] [,LNETNAME=local_netname] [,PERMIT={ ADMIN | SATADM[1] | (ADMIN,SATADM)[1] }] [,QLEV=queue_level_number] [,RNETNAME=remote_netname] ,SESCHA=sescha_name [,STATUS={ ON | OFF }] ***additional operand on BS2000 systems*** [,NETPRIO={ MEDIUM | LOW }]

[1]only permitted on BS2000 systems

lpapname

LPAP partner name, i.e. the logical name of the partner application, which is used by the program units of the local application to address the partner application. *lpapname* applies only in the local application, and can be up to eight characters in length.

The specified name must be unique and must not be assigned to any other object in name class 1. See also section "Uniqueness of names and addresses".

Together with the LTERM names and the OSI-LPAP names, the LPAP names form a common name class.

BUNDLE= master_lpap_name

Name of the master LPAP. If this operand is specified, the LPAP becomes a slave LPAP of an LU6.1-LPAP bundle.

You define the *master_lpap_name* with a MASTER-LU61-LPAP statement.

Messages sent to the master LPAP of an LPAP bundle with an APRO call are distributed to the slave LPAPs of this LPAP bundle by openUTM. This allows the application to distribute the messages to be sent across several partner applications of the same type without the need to program this explicitly.

DEAD-LETTER-Q= specifies whether asynchronous messages to this LPAP partner that could not be sent due to a permanent error are to be saved in the dead letter queue.

Monitoring of the number of messages in the dead letter queue is enabled and disabled with the MAX ...,DEAD-LETTER-Q-ALARM statement.

YES Asynchronous messages to this LPAP partner that could not be sent due to a permanent error are saved in the dead letter queue provided (in the case of message complexes) no negative confirmation job has been defined.

NO No asynchronous messages to this LPAP partner are saved in the dead letter queue.

Default: NO

> Main jobs for message complexes (MCOM) with negative confirmation jobs are never saved in the dead letter queue as the negative confirmation jobs are activated in case of errors.
>
> If the number of messages in the dead letter queue is limited with QLEV, messages may be lost in the event of errors. If the number is not limited, the openUTM page pool generated must be sufficiently large. If there is a threat of a page pool bottleneck, the dead letter queue can be locked during application run with STATUS=OFF.

KSET= keysetname

Name of the key set assigned to the partner application in the local application. The key set is defined using the KSET statement. The partner application can only start those services or address those remote services generated in the local application which are not locked, i.e. for which no lock code has been defined, and whose key codes are defined in the key set *keysetname*.

The local application can thus be secured against unauthorized access by the partner application.

Default: No key set,
i.e. only transaction codes that are not protected with lock codes can be started by the partner application.

LNETNAME= local_netname

This is required only for heterogeneous links. *local_netname* identifies the VTAM name defined for the UTM application in the CICS or IMS partner application.

Default: Blanks

NETPRIO= This operand is only supported on BS2000 systems.

Transport priority to be used on the transport connection assigned to this LPAP partner.

Default: MEDIUM

PERMIT= Authorization level of the partner application

ADMIN The partner application can execute administration functions in the local application.

SATADM The partner application can execute preselection functions in the local application, i.e. it can activate and deactivate the SAT logging of certain events (UTM SAT administration authorization).

(ADMIN,SATADM) The partner application can execute administration and preselection functions in the local application.

Default:
The partner application cannot execute administration functions in the local application.

QLEV= queue_level_number

Maximum number of asynchronous messages that can be accommodated in the message queue of the LPAP partner. If this threshold value is exceeded, further APRO-AM calls to this LPAP partner are rejected with UTM message 40Z.

Default: 32767
Minimum value: 0
Maximum value: 32767 (i.e. unlimited)

RNETNAME= remote_netname

This parameter is required only for heterogeneous links. *remote_netname* identifies the VTAM name of the CICS or IMS partner application.

Default: Blanks

SESCHA= sescha_name

The session characteristics that apply for communication between the local application and the partner application are defined under *sescha_name* in the SESCHA statement (see "SESCHA - define session characteristics for distributed processing based on LU6.1"). By specifying *sescha_name* here, you can assign this set of session characteristics to the LPAP partner.

This is a mandatory operand.

STATUS= Specifies whether the LPAP partner is locked. The status can be changed during operation using the administration command KDCLPAP.

ON	The LPAP partner is not locked. Connections can be established between the partner application and the local application or connections already exist. Default: ON
OFF	The LPAP partner is locked. No connections can be established between the partner application and the local application.

6.5.23 LSES - define a session name for distributed processing based on LU6.1

The LSES control statement required only for communication based on the LU6.1 protocol.

For more information about generating LU6.1 connection see "Distributed processing via the LU6.1 protocol".

It allows you to define a common session name for the connection established between two applications for distributed processing. This name is then used to resume an interruptedcommunication process. LSES also enables you to allocate the session to an LPAP partner. For this purpose, each LPAP statement must be assigned at least one LSES statement. In the case of parallel sessions, several different session names must be defined for the LPAP partner *lpapname*.

An LPAP partner must be always be assigned the same number of sessions (LSES statement) and transport connections (CON statement).

Exception: More LSES statements than CON statements can be assigned to an LPAP partner for a UTM cluster application.

If a session is defined for the local application with LSES AAA, RSES=BBB, this sessionmust be defined with LSES BBB, RSES=AAA in the generation of the partner application.

To ensure that the USER and session name need not be unique in two connected applications, the common session name consists of two parts:
sessionname = local_sessionname + remote_sessionname

LSES	`local_sessionname`
	`  ,LPAP=lpapname`
	`[ ,RSES=remote_sessionname ]`
	addtional operand on Unix, Linux and Windows systems
	`[ ,NODE_NAME=nodename ]`

local_sessionname — Name of the session in the local application.
The specified name must be unique and must not be assigned to any other object in name class 2. See also "Uniqueness of names and addresses".

LPAP= lpapname

Name of the LPAP partner assigned to the partner application.
local_sessionname is used for communicating with the partner application assigned to the LPAP partner *lpapname* in the local application.

NODE-NAME= node_name

This parameter is only relevant for UTM cluster applications on Unix, Linux and Windows systems.

To ensure that openUTM is able to select the "right" session when establishing a session with a partner application, you must assign the LU6.1 sessions to the node applications via the NODE-NAME operand. The following dependencies apply:

- The name specified in NODE-NAME must be defined in a CLUSTER-NODE statement using the identically named NODE-NAME operand.
- The host name for the node (HOSTNAME) specified in this CLUSTER-NODE statement must be referenced in a CON statement in the partner application (PRONAM).
- The LPAP name specified in this CON statement must, in the partner application, be specified in an LSES statement that matches the LSES statement generated here. I.e. the local session name in the partner application corresponds to the RSES name that is specified here and vice versa.

See also "Generation notes" and "Procedure when generating LU6.1 connections".

Default value: eight spaces, i.e. not a node application.

In the case of standalone applications, NODE-NAME may not contain any values other than spaces.

RSES= remote_sessionname

Remote half session name

Default: *remote_sessionname=local_sessionname* is set, if RSES is not named.

6.5.24 LTAC - define a transaction code for the partner application

The LTAC control statement allows you to define a local transaction code for a service, a remote service program or a remote TAC queue in a partner application. LTAC statements can be generated for communication based on both the LU6.1 protocol and the OSI TP protocol.

The local transaction code is assigned either

- the name of a transaction code in a specific partner application (with single-step addressing), in which case the local transaction code addresses both the partner application and the transaction code in this application, or
- the name of a transaction code in any partner application (with double-step addressing). The partner application in which the service program addressed by the local transaction code is to run must be specified explicitly in the program interface.

```
LTAC  ltacname
      [ ,{ ACCESS-LIST=keysetname | LOCK=lockcode } ]
      [ ,LPAP=lpapname ]
      [ ,LTACUNIT=ltacunit ]
      [ ,RTAC={ C'rtacname' |
                rtacname |
                recipient_TPSU_title [ ,CODE={ ST AN D ARD |
                                               PRINTABLE-STRING |
                                               T61-STRING |
                                               INTEGER }
                                     ]
              }
      ]
      [ ,STATUS = { ON | OFF } ]
      [ ,TYPE={ D | A } ]
      [ ,WAITTIME=( time1,time2 ) ]
```

ltacname	Name of a local transaction code defined for the remote service program
ACCESS-LIST=	keysetname ACCESS-LIST= is used to specify the access authorizations that the user of the local UTM application must have in order to be able to send a job to the remote program. Whether the job is actually carried out by the remote application will depend on the access authorizations that are defined there. ACCESS-LIST may not be specified in conjunction with the LOCK=*lockcode* operand. For *keysetname* you must enter the name of a key set. The key set must be defined using a KSET statement. A user can only access the LTAC if the key set of the user (USER ...,KSET=) contains at least one of the key codes contained in the key set (access list) *keysetname* of the LTACs. If you enter neither ACCESS-LIST=*keysetname* nor LOCK=*lockcode* the LTAC is not protected and any user of the local application is able to start the remote service program. Default: no key set
LOCK=	lockcode May not be specified in conjunction with the ACCESS-LIST= operand. LOCK= specifies the Lock code of the remote service program. A service secured with a lock code can only be addressed by a program unit if the program unit was started under a user ID (KCBENID) and from a client or a partner application (KCLOGTER) whose key set contains a key code that matches the lock code. If you enter neither ACCESS-LIST=*keysetname* nor LOCK=*lockcode* the LTAC is not protected and any user of the local UTM application is able to start the remote service program. Default: 0 (no lock code) Maximum value: Value of MAX ...,KEYVALUE=*number*
LPAP=	lpapname This identifies the partner application to which the service program belongs. You must enter the name of the LPAP partner assigned to this partner application or the name of an LU6.1-LPAP bundle or an OSI-LPAP bundle. If the LPAP= operand is not specified, the name of the partner application must be entered in the APRO function call (in the KCPA field).

LTACUNIT= ltacunit

Specifies the number of accounting units that are calculated for each call of this LTAC in the accounting phase of the UTM accounting. The accounting units are added to the accounting unit counter of the user ID that called the LTAC.
You may only specify integer values. This operand is only relevant if you are using the “UTM Accounting” function. Further information on the UTM Accounting can be found in the openUTM manual “Using UTM Applications”.

Default value: 1
Minimum value: 0
Maximum value: 4095

RTAC= Name of the transaction code for the remote service program in the partner application. *ltacname* is used in the local application to address a service program defined under this transaction code (*recipient TPSU-title*) in the partner application.

In the case of asynchronous communication you can also specify the name of a TAC queue in the partner application.

Default: *rtacname=ltacname*

C‘rtacname‘ | rtacname | recipient_TPSU_title The name of the transaction code for the remote service program in the partner application (*recipient_TPS_title*) can be specified in the form of a character string or a number. A character string can be entered in the format *C‘rtacname‘* or *rtacname*.

For *recipient_TPSU_title*, the OSI TP standard distinguishes between the code types printable string, T.61 string, and integer, which are used internally by openUTM to represent the RTAC name.

CODE=

STANDARD If *recipient_TPSU_title* is specified in the form of a character string, it can be up to eight characters in length. It can only contain characters that are permitted for TAC names. Further information can be found in section "Format of names".

CODE=STD must be used for communication based on the LU6.1 protocol, and is recommended if the partner application is a UTM application.

For communication based on the OSI TP protocol, CODE=PRINTABLE- STRING is used internally.

Default: STANDARD

PRINTABLE-STRING

The *recipient_TPSU_title* string can be up to 64 characters in length, and is case-sensitive.

If the partner application is a UTM application, *recipient_TPSU_title* can be up to eight characters in length. It can only contain characters permitted for TAC names. If these requirements are not met, the string can only be used for heterogeneous links based on the OSI TP protocol.

The following characters are permitted for the code type PRINTABLE- STRING:

- A, B, C, . . . , Z
- a, b, c, . . . , z
- 0, 1, 2, . . . , 9

and the following special characters:

apostrophe	’
hyphen	-
blank	<SPACE>
colon	:
question mark	?
equals sign	=
comma	,
plus sign	+
period	.
left parenthesis	(
right parenthesis	)
slash	/

T61-STRING

With the code type T61-STRING, openUTM supports all characters of the code type PRINTABLE-STRING as well as the following special characters:

dollar sign	$
greater than sign	>
less than sign	<
ampersand	&
commercial at	@
number sign	#
semicolon	;
percentage sign	%
asterisk	*
underscore	_

INTEGER

For *recipient_TPSU_title*, you can specify a positive integer between 0 and 67108863.

This is permitted only for partner applications which are not UTM applications and which communicate on the basis of the OSI TP protocol.

STATUS=

This defines whether or not the *ltacname* of the remote service program is locked when the local application is started.

The value entered for STATUS= applies until it is changed using the KDCLTAC administration command.

ON

The transaction code *ltacname* is not locked, i.e. jobs are accepted for the corresponding service program.

Default: ON

OFF

The transaction code *ltacname* is locked, i.e. jobs are not accepted for the remote service program.

TYPE=

This defines whether the remote service program is operated in dialog or asynchronous mode.

D

The remote service program is operated in dialog mode.

Default: D

A

The remote service program or the remote TAC queue is operated in asynchronous mode.

WAITTIME= (time1,time2)

Maximum time spent waiting for a session to be reserved. By appropriately selecting this wait time, you can limit the wait time of a user on the terminal that requests the remote service.

time1 Number of seconds spent waiting for a session to be reserved (possibly including connection setup) or for an association to be established when starting a remote service program.

- *time1* != 0 for asynchronous TACs:
  An asynchronous job is always placed in the message queue of the partner application.
- *time1* != 0 for dialog TACs:
  A dialog job is accepted if a logical connection exists to the partner application.
- *time1* = 0 for asynchronous TACs:
  An asynchronous job (FPUT job) that is not time-driven is only entered in the message queue of the partner application if there is a logical connection to the partner application. If there is no connection, then the FPUT call is rejected with the return code 40Z, KD13.
- *time1*=0 for dialog TACs:
  If there is no session or association generated for which the local application is the contention winner, then the dialog job (APRO DM call) is rejected with 40Z, KD11. If there are sessions/associations for which the local application is the contention winner, but none are free when the program ends, then the transaction is rolled back.

In the case of asynchronous jobs to OSI TP partners *time1* is always set internally to 60 seconds, regardless of the value actually set.

If there is no logical connection to the partner application, then dialog jobs are rejected, regardless of the value of *time1*. The establishment of a connection is initiated at the same time.

time2

Maximum number of seconds spent waiting for a response from the job receiver. This can be used to restrict the wait time for the terminal user.
time2 = 0 means “wait indefinitely”.
time2 is only relevant for dialog LTACs, the wait times for asynchronous LTACs are defined using UTMD ... CONCTIME=(...,*time2*).

> If a value > 0 is specified in *time2* then this value is ignored by openUTM if a KDCSHUT WARN or GRACE has been issued and the local service has initiated the end of the transaction. In this case, openUTM chooses the wait time in such a way that the transaction is rolled back before the application is terminated in order, if possible, to prevent the application from being terminated abnormally with ENDPET.

Default value: WAITTIME = (30,0).
Minimum value: WAITTIME = (0,0)
Maximum value: WAITTIME = (32767,32767)

Wait times can be modified using the UTM administration (e.g. with the KDCLTAC command).

6.5.25 LTERM - define an LTERM partner for a client or printer

The LTERM control statement allows you to define an LTERM partner as the logical access point for a client or printer of the application. Clients are terminals, UPIC clients and transport system applications (DCAM, CMX and socket applications, or UTM applications generated as transport system applications).

LTERM partners are used by clients and printers to establish a connection with the UTM application. They are assigned physical clients or printers using the PTERM statement. You can also define pools of LTERM partners; further information can be found in the description of the TPOOL statement in section "TPOOL - define an LTERM pool".

LTERM partners can also be predefined, i.e. they aren't assigned to a client/printer yet. The LTERM partner -> PTERM assignment can be defined later on during operation using the KDCSWTCH administration command.

A separate LTERM statement must be issued for all clients defined in a PTERM statement.

> Printers are not supported by openUTM on Windows systems.

LTERM	ltermname [,BUNDLE=master-lterm] [,GROUP=primary-lterm] [,KSET=keysetname] [,LOCK=lockcode] [,QAMSG={ **Y**ES | **N**O }] [,QLEV=queue_level_number] [,RESTART={ YES | **N**O }] [,STATUS={ ON | OFF }] [,USAGE={ D | O }] [,USER=username] ***additional operands on BS2000, Unix and Linux systems*** [,CTERM=ltermname2] [,PLEV=print_level_number] ***additional operands on BS2000 systems*** [,ANNOAMSG={ Y | N }] [,FORMAT= { + | * | # }formatname] [,KERBEROS-DIALOG = { **Y**ES | **N** O }] [,LOCALE=([lang_id][,[terr_id][,ccsname]])] [,NETPRIO={ **M**EDIUM | **L**OW }]

> The operands LOCK=, KSET=, USER= and ANNOAMSG= are only valid for clients; the operands CTERM= and PLEV= are only valid for printers.

ltermname

Name of the LTERM partner up to eight characters in length.

ltermname is used

- to assign a client or printer to the LTERM partner in the PTERM statement.
- by the program units of the application to address clients, printers, and other TS applications (not server-to-server communication) assigned to the LTERM partner.

The specified name must be unique and must not be assigned to any other object in name class 1. See also "Uniqueness of names and addresses".

ANNOAMSG=

(**anno**unce **a**synchronous **m**e**s**sa**g**e)

This operand is only supported on BS2000 systems.

This operand applies only for LTERM partners used by terminals (USAGE=D) to sign on to the UTM application.

Y

An asynchronous message to this terminal is announced in advance by outputting UTM message K012 in the system line. The user must then request the message using the UTM command KDCOUT.

Default: Y

N

An asynchronous message to this terminal is sent immediately, i.e. without prior announcement.

BUNDLE=

master-lterm

Name of a master LTERM in a LTERM bundle (connection bundle). By specifying *master-lterm*, this LTERM becomes a slave LTERM of the corresponding connection bundle.

The master LTERM specified here must have been generated in a preceding LTERM statement. Do not assign a PTERM to a master LTERM.

Connection bundles permit load balancing (see "LTERM bundle").

Connection bundles can be generated for APPLI or SOCKET connections (PTYPE operand of the corresponding PTERM statement).

BUNDLE must not be specified together with GROUP or CTERM.

CTERM=

ltermname2

(**c**ontrol **term**inal)

This operand is only supported on BS2000, Unix and Linux systems.

This only needs to be specified for LTERM partners generated for printers (USAGE=O). *ltermname2* is the name of an LTERM partner (up to eight characters in length) which was configured as a printer control LTERM (LTERM ...,USAGE=D). The printer control LTERM can be assigned one or more LTERM partners which were configured for printers. It is used to manage printers, print jobs, and printer queues.

Default: Blanks, i.e. no printer control LTERM

GROUP= primary-lterm

Name of a primary LTERM. By specifying *primary-lterm*, this LTERM becomes an alias LTERM of the corresponding primary LTERM. They define a LTERM group.

In a LTERM group you assign several LTERMs to one connection (see "LTERM groups").

The primary LTERM specified here must have been generated in a preceding LTERM statement. The primary LTERM must be a normal LTERM assigned to a PTERM with PTYPE=APPLI or PTYPE=SOCKET or the master LTERM of a connection bundle. Do not assign a PTERM to an alias LTERM.

> GROUP must not be specified together with BUNDLE or CTERM.

FORMAT= This operand is only supported on BS2000 systems.

Designates the start format of the LTERM partners. Start formats can only be defined for terminals. It only makes sense to specify a start format if the application is generated without user IDs or if you are using your own signon service.

If the LTERM partner is assigned to a UPIC client, then specifying a start format has no effect.

If the application is generated **without** user IDs, this format is output instead of UTM message K001. Following a terminal-specific restart, the start format is not displayed, rather the KDCDISP command is executed.

If the application is generated **with** user IDs, the name of the start format can be queried in the first part of the sign-on procedure using the SIGN ST call. If you do not use your own sign-on procedure, you cannot use the LTERM-specific start format.

The sign of the format consists as follows:

+, * or # followed by an alphanumeric name (*formatname*) up to seven characters in length.

#formats can only be used in the context of a sign-on procedure.

The terms have the following meanings:

+ When the next MGET call of the program unit is issued, each entry in a format field is preceded by 2 bytes for the attribute field in the KDCS message area, i.e. the field properties can be modified by the program unit.
The format name at the KDCS interface is +*formatname*.

* When the next MGET call of the program unit is issued, the entry in a format field is not preceded by any bytes for an attribute field, i.e. the field properties cannot be modified by the program unit. The format name at the KDCS interface is **formatname*.

This identifies a format with extended user attributes. The field properties and global format properties can be modified by the program unit. The format name at the KDCS interface is # *formatname*.

Default: no start format

KERBEROS-DIALOG =

This operand is only supported on BS2000 systems.

YES — A Kerberos dialog is performed when a connection is established for terminals that support Kerberos and that connect to the application directly via this LTERM partner (not via OMNIS).

openUTM stores the Kerberos information in the length resulting from the maximum lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated to this length and stored.

The KDCS call INFO (KCOM=CD) allows a program unit run to read this information unless a user subsequently signs on with an ID card. In this event, the Kerberos information is overwritten by the ID card information.

If the maximum of the lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH is zero, a warning message is issued.

NO — No Kerberos dialog is performed.

Default.

KSET= keysetname

This applies only to clients generated as dialog partners (USAGE=D). *keysetname* is the name of a key set defined using the KSET statement and assigned to the LTERM partner *ltermname*. *keysetname* may be up to 8 characters long.

A maximum of one key set can be assigned to each LTERM partner. This defines the access permissions for this LTERM partner with respect to using the services of the application and remote services (LTACs) generated in this application.

This LTERM partner can only be used to start services of the application that are protected with a lock code or an access list and only address remote services that are protected with a lock code or an access list if the following applies: The key set assigned to the LTERM partner and the KSET of the UTM user ID under which sign-on using this LTERM partner was performed must contain the key code or access code that matches the lock code or access list.
The lock/key code concept and the access list concept are described in detail in the openUTM manual “Concepts und Functions”. An introduction to data access control can be found in section "Lock/key code concept".

Services whose TACs are not secured with codes can be called by the user or the client program without restriction.

In the case of an application in which user IDs have been defined and for which data access control is not required for terminals, you can assign all key codes to the terminals as follows:

```
LTERM ...,KSET=MASTERSET
KSET MASTERSET,KEYS=MASTER
```

Default: No key set

LOCALE= (lang_id,terr_id,ccs_name)

This operand is only supported on BS2000 systems.

Language environment of the client that signs on to the application via this LTERM partner.

lang_id Freely selectable language identifier for the client, up to two characters in length.

The language identifier may be queried by the program units of the application, so that messages can be sent to the terminal in the communication partner's language.

terr_id Freely selectable territorial identifier for the client, up to two characters in length.

The territorial identifier may be queried by the program units of the application, so that messages can be sent to the terminal taking into consideration any special territorial features of the communication partner's language.

ccsname (**c**oded **c**haracter **s**et **name**)
Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). The character set must be compatible with an ISO character set supported by the terminal assigned to this LTERM partner.

During generation, KDCDEF cannot check the validity of the CCS name under the BS2000 system or the compatibility condition. If you specify a CCS name which is not defined in XHCS, this results in a PEND ER when an attempt is made to establish a connection via this LTERM partner during runtime.

The character set with the specified CCS name is used for:

- outputting dialog messages on 8-bit terminals if the application is generated without user IDs or if a user is not signed on to the terminal, and another CCS name is not explicitly selected using an edit profile or a format.
- outputting asynchronous messages on 8-bit terminals if another CCS name is not explicitly selected using an edit profile or a format.

Default: Locale defined in the MAX statement

LOCK= lockcode

This applies only for clients (USAGE=D).
Lock code assigned to the LTERM partner as a logical numerical lock. *lockcode* is a number between 1 and the maximum value permitted by the application (MAX ...,KEYVALUE=*number*). It is only possible to sign on to this LTERM partner under a UTM user ID (USER) for which a key set has been generated with a key code that contains the lock code of the LTERM partner. If the application is generated without user IDs (no USER statement), the LOCK= operand is ignored.

Default: 0, i.e. no lock code
Maximum value: Value of MAX ...,KEYVALUE=*number*

NETPRIO= This operand is only supported on BS2000 systems.

Transport priority to be used on the transport connection assigned to this LTERM partner.

NETPRIO has no significance for LTERM partners that are assigned to a PTERM using PTYPE=SOCKET or PTYPE=*RSO.

Default:
MEDIUM for clients
LOW for printers

PLEV= print_level_number
(**p**rint **lev**el)

This operand is only supported on BS2000, Unix and Linux systems.

This applies only for printers. The PLEV= operand allows you to access printers from various UTM applications (printer sharing). The connection between the UTM application and the printer exists only while the print job is being transferred, thus allowing other applications to establish a connection as required.

This operand defines the number of printer messages at which openUTM attempts to establish a connection with the printer. openUTM continues to collect these messages until the threshold value defined with PLEV= is reached. It then establishes a logical connection to the printer. The connection is shut down again as soon as there are no further messages for this printer. Another application can then output messages to the printer if necessary. If the client is assigned a printer pool, openUTM attempts to establish a connection to all printers in the pool as soon as the threshold value is reached. When all messages have been sent, the connection to all printers in the pool is shut down again.

If the connection to the printer is shut down (e.g. by administration) even through the threshold value PLEV= is still exceeded, openUTM attempts to reestablish the connection at intervals defined in MAX ...,CONRTIME=*time*.

If PLEV=0 is specified, the connection is not shut down even if there are no further output messages.

If PLEV>0 is specified, the operands RESTART=NO or USAGE=D must not be specified for the LTERM partner. On BS2000 systems QAMSG=NO must also not be specified.

> ¡ If PLEV>0 is specified, the operand CONNECT=YES of the associated PTERM statement has no effect.

Default value: 0
Minimum value: 0
Maximum value: 32767
If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.

QAMSG= (**q**ueue **a**synchronous **m**e**s**sa**g**e)

YES An asynchronous message (FPUT job) to the client or printer is buffered by openUTM in the message queue of this LTERM partner, even if the client or printer is not connected to the application.

Default: If RESTART= YES

NO An FPUT job sent to this client or printer is rejected with the return codes KCRCCC=44Z and KCRCDC=K705 if the client or printer is not connected to the application.

Default: If RESTART=NO

QLEV= queue_level_number

(**q**ueue **lev**el)
Specifies the maximum number of asynchronous messages simultaneously buffered by openUTM in the message queue of the LTERM partner. If this threshold value is exceeded, openUTM rejects all further FPUT or DPUT calls for this LTERM partner with 40Z.
QLEV= can be used to control the size of the page pool more effectively. This is because the number of asynchronous messages cannot exceed the *queue_level_number*.

Default value: 32767
Minimum value: 0
Maximum value: 32767 (i.e. unlimited)
If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.

> ¡ You should not specify a QLEV < PLEV for a printer, as this would mean that the connection to this printer would have to be established by administration.

RESTART= Processing of asynchronous messages when the client link is disconnected.

YES When the link to the client assigned to this LTERM partner is disconnected, asynchronous messages are retained. If user IDs are not generated in this application, openUTM performs an automatic service restart for this LTERM partner.

Default: YES

NO When the link to the client assigned to this LTERM partner is disconnected, openUTM deletes all asynchronous messages in the message queue of the LTERM partner. If these are messages of a UTM message complex, the negative confirmation job is activated. It is possible to relieve the load on the page pool by specifying RESTART=NO.

If QAMSG=YES is specified, you must not specify RESTART=NO.

If user IDs are not defined in the application, openUTM does not perform an automatic service restart for clients or printers, i.e.:

- If the connection is shut down by KDCOFF, if it is lost, or if the application is terminated normally, the service is rolled back to the last synchronization point and terminated. The event exit VORGANG is then called with KCKNZVG=D (=Disconnect).
- During a UTM warm start following abnormal termination of the application, an open service for this LTERM partner is terminated without calling the event exit VORGANG.
- Following connection setup, KDCDISP/KDCLAST behaves in the same way as after regeneration, i.e. the UTM message `K020 NO MESSAGE(S) PRESENT` is output.

STATUS= Status of the LTERM partner following connection setup. This can be modified during runtime using the administration command KDCLTERM.

ON — The client or printer assigned to this LTERM partner is not locked, i.e. you can work with it as soon as the connection has been established.

Default: ON

OFF — The client or printer assigned to this LTERM partner is locked.

USAGE= — Type of LTERM partner

D — The LTERM partner is configured as a dialog partner. Both the client and the application can send messages via the connection between the client and the local application.

Default: D

O — The LTERM partner is configured for an output medium. It is only possible to send messages from the application to the printer or TS application etc.

USER= — username

This only applies if the LTERM has been configured as a dialog partner (USAGE=D). Depending on the type of the assigned client, this operand has the following effect:

- If a terminal is assigned to the LTERM partner, openUTM executes an automatic KDCSIGN for the user ID *username* when establishing a logical connection between the client assigned to this LTERM partner and the UTM application. Note that when the automatic KDCSIGN is used, access protection is limited.
  You should not specify this operand unless you are certain that an authorized user is working under this user ID at this client. After the logical connection is set up, the client is in the same state as if the user *username* executed the KDCSIGN command (BS2000 systems) or signon check (Unix, Linux and Windows systems) successfully. The user ID must be defined using the USER statement.
- If the LTERM partner is assigned a client of type APPLI or UPIC, then the user ID *username* is reserved for this client (as the connection user ID). The client is signed on under this user ID when the connection is established. Another client or terminal user cannot sign on to the UTM application with this user ID.
  If a user ID with the name of the LTERM partner was generated explicitly by means of a USER statement, openUTM assigns this user ID exclusively to the LTERM partner. If the LTERM partner is to send administration calls to the application, then a user ID with administration authorization must be specified if the client is not signed-on using a real user ID. If transaction codes are to be called from this client that are protected by lock codes, then this user ID must be assigned an appropriate key set.
  If no user ID was specified, then KDCDEF implicitly creates a user ID with the name of the LTERM partner and the value defined in LTERM ...,RESTART=.

Default: No automatic KDCSIGN

6.5.26 MASTER-LU61-LPAP - define the master LPAP of an LU6.1-LPAP bundle

The MASTER-LU61-LPAP statement allows you to specify the name and properties of a master LPAP for an LU6.1 LPAP bundle.

Slave LPAPs are assigned to a master LPAP of an LU6.1 LPAP bundle with the BUNDLE parameter of the LPAP statement. The master LPAP and the slave LPAPs together form an LPAP bundle. LPAP bundles allow messages to be distributed automatically across several LPAP partners (see "LU6.1-LPAP bundles").

`MASTER-LU61-LPAP`	`master_lpap_name` `[ ,STATUS={ ON | OFF } ]`

master_lpap_name

Name of the master LPAP of an LU6.1 LPAP bundle. This name is only of significance in the local application and must differ from the names of LTERMs, LPAPs, OSI-LPAPs and TACs defined in this application.

master_lpap_name can be up to 8 characters in length.

STATUS= Specifies whether the MASTER-LU61-LPAP is locked.

ON The MASTER-LU61-LPAP is not locked.

OFF The MASTER-LU61-LPAP is locked. Jobs for the MASTER-LU61-LPAP are rejected.

6.5.27 MASTER-OSI-LPAP - defining the master LPAP of an OSI-LPAP bundle

WIth the control statement MASTER-OSI-LPAP you specify the name and properties of a master LPAP for an OSI-LPAP bundle.

A master LPAP of an OSI-LPAP bundle is assigned slave LPAPs with the BUNDLE parameter of the OSI-LPAP statement. The master LPAP and the slave LPAPs together form a LPAP bundle. LPAP bundles allow messages to be distributed automatically across several LPAP partners (see "OSI-LPAP bundles").

`MASTER-OSI-LPAP`	`master_lpap_name`
	`,APPLICATION-CONTEXT=context_name`
	`[ ,STATUS={ ON | OFF }]`

master_lpap_name

Name for the master LPAP in an OSI-LPAP bundle. This name is only significant in the local application. It must be different from the names of the LTERMs, LPAPs, OSI-LPAPs, and TACs defined in the application.
master_lpap_name can be a maximum of 8 characters long.

APPLICATION-CONTEXT=context-name

Name of the application context to be used for communication with the remote partner.

All slave LPAPs in an OSI-LPAP bundle must be assigned to the same application context as the master LPAP.

STATUS= Specifies whether the MASTER-OSI-LPAP is locked.

ON The MASTER-OSI-LPAP is not locked.

OFF The MASTER-OSI-LPAP is locked. Jobs for the MASTER-OSI-LPAP are rejected.

6.5.28 MAX - define UTM application parameters

- The MAX control statement allows you to define the maximum values, timers, process values and system parameters of a UTM application. For instance, these include:
- the name of the application
- the base name or the base directory for UTM files
- single or dual-file operation of the KDCFILE
- size of a UTM page (block size of UTM storages and buffers)
- the maximum number of
  - processes
  - key codes
  - GSSBs
  - LSSBs
  - UTM pages in the buffer for user log records, etc.
- threshold values for monitoring the size of SYSLOG file generations if the SYSLOG is created as an FGG (SYSLOG-SIZE operand)
- the default language environment of the UTM application (LOCALE operand)
- whether or not SM2 can be used for performance monitoring in the application

The parameters of the MAX control statement can be split into several MAX statements. If the same operand is inadvertently entered in several MAX statements, the first value entered for this operand is taken as valid.

Mandatory operands:

APPLINAME=, KDCFILE= and TASKS=.

Additional mandatory operands on Unix, Linux and Windows systems:

SEMKEY= or SEMARRAY= (semaphore keys), IPCSHMKEY=, KAASHMKEY= and CACHESHMKEY=.

In OSI TP applications additionally: XAPTPSHMKEY= and OSISHMKEY=.

The mandatory operands need to be defined once.

Note on UTM cluster applications on Unix, Linux and Windows systems:

If you modify one of the operands APPLINAME, APPLIMODE, GSSBS, LSSBS, KB or NB then you must regenerate both the initial KDCFILE and the UTM cluster files by specifying GEN=(CLUSTER,KDCFILE) in the OPTION statement.

For clarity, all operands of the MAX statement are listed in a table following the operand descriptions.

Operands valid for all operating systems

```
MAX  [ APPLIMODE={ SECURE | FAST } ]
       ,APPLINAME=appliname
     [ ,ASYNTASKS={ atask_number | (atask_number,service_number) } ]
     [ ,BLKSIZE={ 2K | 4K | 8K } ]
     [ ,CACHESIZE=( number,paging,{ NORES | RES }[1]  { ,PS | DS }[1]  ) ]
     [ ,CLRCH={ c | C'c'| X'xx' } ]
     [ ,CONN-USERS=number ] (mandatory on Unix, Linux and Windows systems)
     [ ,CONRTIME=time ]
     [ ,DATA-COMPRESSION={ STD | YES | NO } ]
     [ ,DEAD-LETTER-Q-ALARM=number ]
     [ ,DESTADM=destination ]
     [ ,DPUTLIMIT1=( day,hour,minute,second ) ]
     [ ,DPUTLIMIT2=( day,hour,minute,second ) ]
     [ ,GSSBS=number ]
     [ ,HOSTNAME=name ]
     [ ,KB=length ]
       ,KDCFILE=( filebase [, { SINGLE | DOUBLE } ] )
     [ ,KEYVALUE=number ]
     [ ,LEADING-SPACES={ NO | YES } ]
     [ ,LPUTBUF=number ]
     [ ,LPUTLTH=length ]
     [ ,LSSBS=number ]
     [ ,MOVE-BUNDLE-MSGS={ YES | NO } ]
     [ ,NB=length ]
     [ ,NRCONV=number ]
     [ ,OSI-SCRATCH-AREA=value ]

     [ ,PGPOOL=( number,warnlevel1,warnlevel2 ) ]
     [ ,PGPOOLFS=number ]
     [ ,PGWTTIME=time ]
     [ ,PRIVILEGED-LTERM = <lterm-name> ]
     [ ,QTIME = (qtime1,qtime2)]
     [ ,RECBUF=( number,length ) ]
     [ ,RECBUFFS=number ]
     [ ,REDELIVERY=(number1, number2) ]
     [ ,RESWAIT={ time1 | ( time1, time2 ) } ]
     [ ,SM2={ NO | OFF | ON } ]
     [ ,SPAB=length ]
     [ ,STATISTICS-MSG={ NONE | FULL-HOUR } ]
     [ ,SYSLOG-SIZE=size ]
     [ ,SYSTEM-TASKS={ *STD | number } ]
       ,TASKS=number
     [ ,TASKS-IN-PGWT=number ]
     [ ,TERMWAIT=time ]
     [ ,TRACEREC=number ]
     [ ,TRMSGLTH=length ]
     [ ,USLOG={ SINGLE | DOUBLE } ]
```

further operands for BS2000 systems

```
[ ,BRETRYNR=number ]
[ ,CARDLTH=length]
[ ,CATID=( catalog_A,catalog_B ) ]
[ ,LOCALE=( [ lang_id ][,[ terr_id ][ ,ccsname ] ] ) ]
[ ,LOGACKWAIT=time ]
[ ,MP-WAIT=number ]
[ ,PRINCIPAL-LTH=length ]
[ ,REQNR=number ]
[ ,SAT={ ON | OFF } ]
[ ,VGMSIZE=number ]
```

further operands for Unix, Linux and Windows systems

```
  ,CACHESHMKEY=number
  ,IPCSHMKEY=number
[ ,IPCTRACE=number ]
  ,KAASHMKEY=number
  ,OSISHMKEY=number          only mandatory if you generate OSI TP partners
  ,{ SEMARRAY=( number,number1 ) | SEMKEY=( number,...) }
  ,XAPTPSHMKEY=number        only mandatory if you generate OSI TP partners
```

[1]NORES | RES and PS | DS only permitted on BS2000 systems. The long form PROGRAM-SPACE or DATA-SPACE can also be specified instead of PS or DS.

APPLIMODE= This specifies whether the application is a UTM-S or UTM-F application.

SECURE — The application is generated as a UTM-S application.

With UTM-S, openUTM logs all user data so that this data is retained after the application is terminated or following a system crash. In the event of errors, UTM-S guarantees the integrity and consistency of the application data. If a UTM-S application is terminated abnormally, an automatic restart is automatically performed. For this purpose, this variant of openUTM logs all modifications at the end of transactions.

Default: SECURE

FAST — The application is generated as a UTM-F application.

UTM-F offers enhanced performance by eliminating the disk input/output operations performed by UTM-S when logging user and transaction data. With a standalone UTM-F application, openUTM only logs user passwords and changes to the configuration which were made by means of dynamic administration. These modifications are thus retained for the next application run. However, UTM-F applications do not log changes to the user data. They are therefore suitable only for installations in which performance is the most important criterion and the restart facility is not required. This applies in the case of pure information systems, or if all logging functions can be provided by the database system used.

In UTM cluster applications, user data that is valid globally in the cluster is also saved for UTM-F.

APPLINAME= applinname

Name of the UTM application up to eight characters in length. *applinname* defines a transport system access point via which connections to the UTM application can be established.

This is a mandatory operand.

If several application names are required, for example, for distributed processing based on the LU6.1 protocol, these can be assigned to the application using the BCAMAPPL statement. With APPLINAME= you define the primary application name.

applinname must be unique within the local system and must not begin with the character '$'.

BS2000 systems:

This name is subject to the name conventions for BCAM applications. The transport system access point defined with *applinname* supports the transport protocol NEA.

applinname must not begin with a number or with '$' as this is prohibited by BCAM and the application cannot be started otherwise. Please note that KDCDEF cannot intercept numbers.

Unix, Linux and Windows systems:

applinname must be specified when establishing a connection from the terminal (dialog terminal process).

If connections are to be established with partner applications using the application name defined with APPLINAME=, you must also issue an appropriate BCAMAPPL statement (see "BCAMAPPL - define additional application names").

ASYNTASKS= (atask_number,service_number)

Maximum number of resources that may be reserved to process asynchronous jobs.

atask_number Maximum number of processes (BS2000 tasks or work processes on Unix, Linux and Windows systems) of the application which can simultaneously handle jobs with asynchronous transaction codes. This operand allows you to prevent long-running asynchronous processes from affecting dialog operation.

If ASYNTASKS=0, asynchronous TAC classes cannot be generated.

Default: 1
Minimum value: 0
Maximum value: TASKS -1

service_number

Maximum number of asynchronous services that may be open at the same time.

You should set *service_number* to be larger than *atask_number* when one of the two following cases can arise:

- Process switch while processing an asynchronous service:If an asynchronous service consists of several program units and if the transaction code of a follow-up program (follow-up TAC after PEND PA/PR or PEND SP) is located in a different TAC lass than the calling program unit or the priority control is generated for TAC classes (TAC-PRIORITIES statement), then a process switch can occur during processing. The asynchronous service is inactive at first and does not allocate a UTM process, although it remains open.
- Dialogs initiated by synchronous services with LU6.1 or OSI TP partner applications: If a dialog is initiated with a partner application within an asynchronous service (with APRO DM) and it must wait for a response from the partner (via PEND KP or PEND RE), then the asynchronous service remains open until the response arrives (or until a timeout), but it does not allocate a UTM process.

If these cases arise in the application and the value of *service_number* is too small, then the asynchronous processing may be temporarily blocked because *service_number* of inactive services already exist. New asynchronous services cannot be started although no UTM processes are processing asynchronous services at this time.

Default: *atask_number*
Minimum value: *atask_number*
Maximum value: 32767

BLKSIZE=

Size of a UTM page

Please note that, depending on the BLKSIZE specification, each user storage area occupies at least 2K, 4K or 8K in the page pool.

You can only specify BLKSIZE=4K or 8K for UTM cluster applications.

Default

- standalone UTM applications: 2K
- UTM cluster applications that run on 32-bit systems: 4K
- UTM cluster applications that run on 64-bit systems: 8K

Possible values: 2K, 4K, 8K

> ¡ On BS2000 systems you must specify BLKSIZE=4K or 8K
>
> - if the KDCFILE and the USLOG file are created on NK4 disks, or
> - if the KDCFILE is to be used as a Hiperfile (high-performance file).

BRETRYNR= number

This operand is only supported on BS2000 systems.

Number of attempts made by openUTM to transfer a message to the transport system (BCAM) if BCAM cannot accept the message immediately. If this number is exceeded, the connection to the dialog partner is shut down.

BRETRYNR is irrelevant for asynchronous messages output to a dialog partner with PTYPE=APPLI (PTERM statement). If such a message from the transport system is rejected due to a temporary bottleneck, then openUTM releases the process, but does not clear down the connection. After waiting for three seconds, openUTM makes up to three attempts to transfer the message to BCAM. If after the third attempt BCAM still cannot accept the message, then openUTM waits for 3 more seconds before it makes another three attempts to send the message to BCAM. If still unsuccessful, it waits another 3 seconds before making another three attempts, and so on.

Default: 10
Minimum value: 1
Maximum value: 32767 (theoretical value)

CACHESHMKEY= number

This operand is only supported on Unix, Linux and Windows systems.

Authorization key for the shared memory segment containing the global buffer for file access. Keys are global parameters under the Unix, Linux and Windows systems. You cannot specify more than one key. You must enter a decimal number for *number*.

This is a mandatory operand.

CACHESIZE= (number,paging,NORES or RES, PS or DS)

(NORES, RES, PS, DS only on BS2000 systems PROGRAM-SPACE or DATA-SPACE can also be specified instead of PS or DS)

This specifies the size and properties of the cache memory (further information can be found in the openUTM manual “Concepts und Functions”). The values entered here affect the performance of your UTM application.

number

Number of UTM pages in the cache. The size of each UTM page is defined in the BLKSIZE= operand. The cache is used for accessing the page pool, i.e. all input and output operations involving LSSBs, GSSBs, TLSs, LPUT messages, FPUT messages, MPUT messages to clients, and some types of UTM administrative data. Data is not written to the KDCFILE until the cache becomes full or the transaction is terminated.

KDCDEF rounds up this number to a multiple of 32.

Default value:
1024 (corresponds to 2, 4 or 8 MB, depending on the value of BLKSIZE=)
Minimum value:
32 (corresponds to 64, 128 or 256 KB, depending on the value of BLKSIZE=)
Maximum value:
Depends on the hardware and operating system, but not larger than 16777184.

> **i** If the cache on BS2000 systems is located in the program space (PS), the cache is created in a common memory pool whose size is always a multiple of 1 MB. The BS2000 system automatically rounds the value specified in CACHESIZE. CACHESIZE should be requested in multiples of 1 MB so that address space is not wasted.

paging

Percentage of cache pages to be written to the KDCFILE in a single batch in the event of a bottleneck, thereby freeing space in the cache. This must correspond to at least eight pages. The value specified here can be modified using the administration command KDCAPPL CACHE=*%_utm_pages*.

Default value: 70(%)
Minimum value: 0, i.e. eight pages are swapped out
Maximum value: 100 (%)

NORES

This operand value is only supported on BS2000 systems.

The cache is created as non-resident.

Default: NORES

RES

This operand value is only supported on BS2000 systems.

The cache is created as resident.

A resident cache can enhance the performance of the UTM application. RES may not be specified together with DATA-SPACE.

Resident cache offers enhanced performance in productive mode, as the cache paging algorithm is designed specifically for use with this type of cache.

> **i** The number of resident pages used in the creation of a resident cache cannot be checked using the COREBIAS operand of the BS2000 command BIAS.

PS or PROGRAM-SPACE

This operand value is only supported on BS2000 systems.

The UTM cache is created in the program space.

Default: PS

DS or DATA-SPACE

This operand value is only supported on BS2000 systems.

The UTM cache is created in one or more data spaces.

If the generated UTM cache is larger than 2GBt, UTM will distribute the cache over more than one data space as a data space may be at most 2GB in size.

The option of creating the UTM cache in a data space should be chosen only if a very large UTM cache is required and the address space (program space) is not sufficient for this.

Use of a data space for the ITM cache always entails a slight loss of performance as a result of the way in which a program can access data in a data space. For applications which require a large UTM cache, these performance disadvantages are, however, counterbalanced by the advantages which a large cache brings through the reduction of file IOs.

Above all for UTM applications it can be advantageous to create a very large UTM cache in a data space. In the case of UTM-F, cache buffers are written to file only in the event of a cache bottleneck. If the UTM cache is generated large enough all file IOs to this page pool may be omitted for such applications.

The maximum size of a UTM cache in data spaces is 8 GB. In other words:

- when BLKSIZE is 2K, the maximum value for *number* is 4,194,304,
- when BLKSIZE is 4K, the maximum value for *number* is 2,097,152,
- when BLKSIZE is 8K, the maximum value for *number* is 1,048,576.

DATA-SPACE may not be specified together with RES.

CARDLTH= length

This operand is only supported on BS2000 systems.

Length of the ID card information in bytes. If the ID card reader is used for sign-on, openUTM stores the ID card information in the length resulting from the maximum of the length specified here and the value generated for MAX PRINCIPAL-LTH. If the information on the ID card is longer, it is truncated and stored in this length.

The KDCS call INFO (KCOM=CD) enables a program to read this information.

CARDLTH must be big enough to ensure that the following applies for all USER statements with
USER ..., CARD = (*pos*, *string*):
pos + length (string) -1 <= CARDLTH.

Default: 0
Maximum value: 255

When a value > 255 is specified, 255 is assumed.
No warning message is output.

CATID= (catalog_A,catalog_B)

This operand is only supported on BS2000 systems.

Catalog IDs to which your KDCFILE is assigned.

If you work with CATIDs, enter the base name without the CATID in KDCFILE= *filebase* (see "MAX - define UTM application parameters").

In the case of single-file operation of the KDCFILE, specify the CATID to which the KDCFILE is to be assigned in *catalog_A*. In this case, *catalog_B* is not specified.

In the case of dual-file operation of the KDCFILE (see "The KDCFILE"), you can assign files with the suffix A to CATID *catalog_A* and files with the suffix B to *catalog_B*. If you only specify a value for *catalog_A*, both files are assigned to this CATID.

CLRCH= Character with which the KB program area and the standard primary working area are overwritten at the end of a dialog step. Possible entries are:

c
C’c’
X’xx’

Where *c* is an alphanumeric character and *x* a hexadecimal character.

Default:
The communication area and standard primary working area are not overwritten.

CONN-USERS= number

This operand is used to control the load on the application. It defines the maximum number of users that can work simultaneously with the application. In the case of an application for which user IDs have not been generated, CONN-USERS= can be used to define the maximum number of clients that can sign simultaneously on to the application via LTERM partners.

- CONN-USERS < number of users/clientsThis prevents all users/clients from working simultaneously with the application.
- CONN-USERS=0The number of simultaneously active users/clients is unrestricted.
- CONN-USERS > number of users/clientsThe application load is not controlled. CONN-USERS= is ignored.

User IDs and clients generated with administration authorization can sign on to the UTM application, even if the maximum number of simultaneously active user IDs has already been reached.

Default value on BS2000 systems: 0 (i.e. no restriction)
Minimum value: 0
Maximum value: 500000

> CONN-USERS is a mandatory operand on Unix, Linux and Windows systems. Please note that *number* cannot be set to a higher value that the number of concurrent user licenses obtained.

CONRTIME= time

(**con**nection **r**equest **time**) Time in minutes after which openUTM retries to establish a connection after failing to establish a connection generated to be established automatically.
If CONRTIME > 0 then, following a disconnection, openUTM first attempts to reestablish the connection immediately and then at the intervals specified in CONRTIME. This applies to the following partners:

- TS applications (PTYPE=APPLI or PTYPE=SOCKET) which openUTM generates with automatic connection setup (PTERM ...,CONNECT=YES,) provided that the connection was not terminated by an administration command or due to the IDLETIME timer running down (see the PTERM statement on "PTERM - define the properties of a client/printer and assign an LTERM partner") .
- OSI TP or LU6.1 partner applications which were generated with automatic connection setup, provided that the connection was not terminated by an administration command or because of the expiry of an IDLETIME timer.
- OSI TP partner to which the asynchronous messages were sent and with which no connection existed at the creation time of the messages.
- On BS2000 systems
  - Printers to which openUTM establishes a connection as soon as the number of print jobs for this printer exceeds the generated threshold value (LTERM ...,PLEV>0). On disconnection, the number of print jobs must be greater than or equal to the threshold value if openUTM is to attempt to re-establish the connection. If CONRTIME!=0, openUTM also attempts to re-establish the connection if this was previously explicitly disconnected using an administration command.
  - Printers to which openUTM automatically establishes a connection (PTERM ...,CONNECT=YES), provided that the connection was not terminated by an administration command.
  - Message distributor (MUX) to which openUTM automatically establishes a connection on start-up, provided that the connection was not terminated by an administration command.

If a connection to this partner is not established when the application is started or the administration command KDCPTERM or KDCLPAP is issued, openUTM attempts to reestablish the connection at intervals specified in CONRTIME=.

If CONRTIME=0, openUTM does not make any attempt to set up the connection.
Exception: A wait time of 10 minutes is set for asynchronous messages to OSI TP partners.

Default: 10 min.
Maximum value: 32767 min.

DATA-COMPRESSION=

This parameter enables data compression to be permitted or not permitted.
If data compression is permitted and enabled, for data of the secondary storages and long-term storages (GSSB, LSSB, TLS and ULS) and the communication area-program area UTM performs data compression in order to reduce the space required for these areas by at least one UTM page. For UTM-S applications in particular this can have a positive effect on the performance because as a result execution in UTM and the file IOs to the page pool are optimized.
UTM attempts to compress the user data only if this enables at least one UTM page to be saved, i.e. only for data spaces which are written with a length of more than one UTM page.

YES

Data compression is permitted and enabled. Data compression can be disabled using administration facilities, e.g. by means of KDCAPPL.

NO

Data compression is not permitted and is disabled. This setting cannot be modified using administration facilities.

STD

Data compression is permitted and enabled for UTM-S applications (APPLIMODE=S).
Data compression is permitted but disabled for UTM-F applications (APPLIMODE=F).

This default setting can be modified using administration facilities.

Default: STD

> The average value for UTM pages saved per data compression can be queried using the administration functions, e.g. using the KDCINF STAT command (see openUTM manual “Administering Applications”) or using WinAdmin or WebAdmin.

DEAD-LETTER-Q-ALARM

Controls monitoring the number of messages in the dead letter queue.

The K134 message is output each time the threshold is reached. for this message the destination MSGTAC can be defined in order to automate handling of the dead letter queue.

Default: 0, monitoring is disabled.
Maximum value: 65535

DESTADM=

destination

Destination to which openUTM sends the results of administration calls processed asynchronously. For *destination*, you can specify:

- an LTERM partner Exception: UPIC-LTERM partners are not permitted!
- the TAC of an asynchronous program
- the TAC queue (type=Q).

Default: Blanks, i.e. no destination; the results are thus lost.

DPUTLIMIT1= (day,hour,minute,second)

Defines the latest possible execution time of a job. Can be specified in relative or absolute time:

time of execution < time of DPUT call + DPUTLIMIT1

The following applies for time specifications in DPUTLIMIT1:

day
Maximum value: 364
Minimum value: 0

hour
Maximum value: 23
Minimum value: 0

minute
Maximum value: 59
Minimum value: 0

second
Maximum value: 59
Minimum value: 0

Default value: DPUTLIMIT1 (360, 0, 0, 0) = 360 days
Default value: DPUTLIMIT2 (1, 0, 0, 0) = 1 day
Minimum value: (0, 0, 0, 0)
Maximum value: (364, 23, 59, 59)

The following must apply for the DPUTLIMIT1 and DPUTLIMIT2 operands:

DPUTLIMIT1 + DPUTLIMIT2 <= (364, 23, 59, 59) < 365 days

i.e. if you enter (364, 23, 59, 59) for DPUTLIMIT1, you must specify DPUTLIMIT2= (0, 0, 0, 0).

DPUTLIMIT2= (day,hour,minute,second)

The time specification for the DPUT call does not contain a number for the year. Furthermore, the desired execution time may already have passed if the DPUT call was delayed.

For this reason, you must decide whether the execution time of a job with an absolute time specification should be attributed to the past, current, or next year.

Since DPUTLIMIT1 + DPUTLIMIT2 must be < 1 year, only one of these three alternatives will be in the permissible open time period (call time - DPUTLIMIT2, call time + DPUTLIMIT1):

- If the only alternative allowed is before the call time, then the DPUT is handled as an FPUT and executed as soon as possible.
- If the only alternative allowed is after the call time, then the DPUT is saved and only converted to an FPUT and executed at the alternative time.
- If none of the three alternatives are in the permissible time period, then the DPUT is rejected.

DPUTLIMIT2 therefore allows you to backdate the specified execution time into the past for time-driven jobs with absolute time specifications. You cannot backdate jobs with relative time specifications.

DPUTLIMIT1 restricts the predating of jobs with absolute or relative time specifications into the future only.

Example 1

```
DPUTLIMIT1 = (300,0,0,0)
DPUTLIMIT2 = (010,0,0,0)
```

The DPUT call time is (005,0,0,0). The current and last years are not leap years.

- DPUTs with relative time (000,0,0,0) to (299,23,59,59) are accepted.
- DPUTs with absolute times (001,0,0,0) to (005,0,0,0) and (360,0,0,1) to (365,23,59,59) are handled as FPUT.
- DPUTs with absolute time (005,0,0,1) to (304,23,59,59) are handled as DPUT.

DPUTs with absolute time (305,0,0,0) to (360,0,0,1) are rejected.

```
-|---------------|------------ ... --------------|-----------|
360             5                               305         360
 FFFFFFFFFFFFFFFFDDDDDDDDDDDDD ... DDDDDDDDDDDDDDD
```

Example 2

- DPUTLIMIT1 and DPUTLIMIT2 are defined exactly as in *Example 1* , but the DPUT call time is (360,0,0,0).
- DPUTs with relative time (000,0,0,0) to (299,23,59,59) are accepted.
- DPUTs with absolute time (350,0,0,1) to (360,0,0,0) are handled as FPUT.
- DPUTs with absolute time (001,0,0,0) to (294,23,59,59) and (360,0,0,1) to (365,23,59,59) are handled as DPUT.
- DPUTs with absolute time (295,0,0,0) to (350,0,0,0) are rejected.

```
-|---------------|------------- ... -----------|-------------|
350             360                           295           350
 FFFFFFFFFFFFFFFFDDDDDDDDDDDDD ... DDDDDDDDDDD
```

The default values are listed under the description of the DPUTLIMIT1 operand.

GSSBS= number

Maximum number of GSSBs (global secondary storage areas) that can exist simultaneously in the application.

Default: 32
Minimum value: 0
Maximum value: 30000

HOSTNAME= name

BS2000 systems:

Name of the virtual host on which the UTM application runs (from the point of view of BCAM). This virtual host must also be generated in BCAM. The name can be up to 8 characters in length. Default value: 8 blanks, i.e the applications runs under the real host.

Unix, Linux and Windows systems:

Can only be specified in standalone applications.

In UTM cluster applications, you can specify a virtual host name in the VIRTUAL-HOST parameter of the CLUSTER-NODE statement.

Name of the host that is specified as the sending address when a connection is established from the UTM application end. HOSTNAME= is required in cluster systems that use the “relocatable” IP address as the sending address and not the stationary IP address.
The name can be up to 64 characters in length.

Default: Blanks, the default processor name of the transport system is used as the sending address.

IPCSHMKEY= number

This operand is only supported on Unix, Linux and Windows systems.

Authorization key for the shared memory segment, which is used for communication between work processes on one side and the dialog terminal or printer processes and the timer process (external processes of an application) on the other side. Keys are global parameters under the Unix, Linux and Windows systems. You cannot specify more than one key. You must enter a decimal number for *number*.

This is a mandatory operand.

IPCTRACE= number

This operand is only supported on Unix, Linux and Windows systems.

In test mode (startup with TESTMODE=ON, see openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”), openUTM writes entries in the trace area of the IPC (shared memory segments for interprocess communication). These entries contain internal information which is required for diagnostic purposes. Each entry occupies 32 bytes. IPCTRACE defines the number of entries in the trace area. If this number is exceeded, openUTM overwrites the existing entries, beginning with the oldest entry.

Default: 1060
Minimum value: 1
Maximum value: 32500

KDCDEF automatically resets values < 1 or > 32500 to the minimum or maximum value without outputting a UTM message.

KAASHMKEY= number

This operand is only supported on Unix, Linux and Windows systems.

Authorization key for the shared memory segment containing the global data. Keys are global parameters under the Unix, Linux and Windows systems. You cannot specify more than one key. You must enter a decimal number for *number*.

This is a mandatory operand.

KB= length

Length of the communication area (KB) in bytes, excluding the KB header and KB return area.

Default: 512
Minimum value: 0
Maximum value: 32767

KDCFILE=

filebase

Base name of the KDCFILE, the user log file, and the system log file SYSLOG. The name entered here must also be specified in the start parameter FILEBASE=filebase when starting the application program (see openUTM manual “Using UTM Applications”).

This is a mandatory operand.

BS2000 systems:
If you use the CATID= parameter to assign catalog IDs to your KDCFILE, the base name must be specified without a CATID. (see section "BS2000 systems:" (The KDCFILE) for the format and length of the name).

Unix, Linux and Windows systems:
filebase is the name of the directory containing the KDCFILE and all application files. This directory must be created before the KDCDEF run.
filebase can be fully or partially qualified and can be a maximum of 29 characters in length for standalone applications, irrespective of whether the name is fully or partially qualified.
filebase can be a maximum of 27 characters in length for UTM cluster applications.

SINGLE

Single-file operation is activated for the KDCFILE.

If the KDCFILE is split (see section "Splitting the KDCFILE"), all KDCFILE files are subject to single-file operation.

Default: SINGLE

DOUBLE

For security reasons, dual-file operation is activated for the KDCFILE.

If the KDCFILE is split (see section "Splitting the KDCFILE"), all KDCFILE files are subject to dual-file operation.

In UTM cluster applications, only SINGLE may be specified.

KEYVALUE= number

Value of the highest key code of the application, and thus the value of the corresponding highest lock code that can be assigned to a transaction code or a terminal for data access control. The operand KEYVALUE=number can also be used to define the maximum number of key codes per key set. openUTM uses this information to optimize the key set tables. You can define up to 4000 key and lock codes. Only numerical lock codes can be defined.

Default: 32
Minimum value: 1
Maximum value: 4000

Exceptions:

- Maximum value (32-bit Unix, Linux and Windows systems): 1976 for MAX ..., BLKSIZE=2K
- Maximum value (64-bit Unix, Linux and Windows systems): 3900 for MAX ..., BLKSIZE=4K

If you enter a value < 1, KDCDEF automatically sets KEYVALUE=1 without outputting a UTM message.

LEADING-SPACES= Specifies how the leading spaces in a messages from a terminal or from a TS application (PTERM ... PTYPE=APPLI or SOCKET) are to be handled.

YES When calling a program unit, leading blanks in messages are passed on to the program unit. The same applies for messages sent to a client with PTYPE=APPLI. A blank acting as a separator between TAC and message is removed if the TAC name < 8 characters.

NO Leading blanks are suppressed.

Default: NO

LOCALE= (lang_id,terr_id,ccsname)

This operand is only supported on BS2000 systems.

Default language environment of the UTM application (see also section“UTM messages”).

> The locale generated here is assigned to all user IDs and clients that sign on via LTERM partners or LTERM pools as the default language environment. This default setting applies unless another locale is explicitly defined for these objects in the corresponding USER, LTERM, or TPOOL statements.

The message module whose language and territorial identifiers match the specifications in the MESSAGE ...LOCALE= and MAX ...,LOCALE= statements becomes the application message module. openUTM sends messages to the message destinations SYSOUT, SYSLST, and CONSOLE from this application message module. The specifications in the application message module also determine the destination of a particular message.

lang_id

Freely selectable language identifier for the UTM application up to two characters in length.

Default: Blanks

terr_id

Freely selectable territorial identifier up to two characters in length.

Default: Blanks

ccsname

(**c**oded **c**haracter **s**et **name**)
Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). During generation, openUTM cannot check whether this condition is fulfilled. KDCDEF will thus accept CCS names to which no character set is assigned.

Default: Blanks, i.e. 7-bit mode

LOGACKWAIT= time

The maximum length of time in seconds that openUTM is to wait for an acknowledgment from an output device. This acknowledgment is

- for a printer, the logical print acknowledgment from the printer,
- for an RSO printer, the acknowledgment from RSO,
- for an FPUT call to another application, the transport acknowledgment.

If confirmation does not arrive within this period, e.g. because the printer has run out of paper, openUTM shuts down the logical connection to the device.

Default: 600
Minimum value: 10
Maximum value: 32767

LPUTBUF= number

Size of the LPUT buffer in UTM pages. The LPUT buffer of the KDCFILE is used to temporarily store LPUT data. This data is not copied to the user log file until the value specified in *number* is exceeded. The user log file USLOG is open only during this copy process.

Default: 1
Minimum value: 1
Maximum value: 1000
KDCDEF automatically resets values > 1000 to 1000 without outputting a UTM message.

> **CAUTION!**
> This operand must be set > 1 if the application contains LPUT calls. Otherwise, the copy process will be started too often. This involves opening and closing the user log files.
> The value entered in LPUTBUF must be selected such that the buffer can accommodate the longest LPUT record. The following must apply:
>
> LPUTBUF * UTM page size >= LPUTLTH + length of KB header (84 bytes)

LPUTLTH= length

Maximum length of the user data in LPUT records in bytes (excluding the KB header).

The maximum length of an LPUT record in the user log file is calculated as follows (see also the openUTM manual „Programming Applications with KDCS”, user log file):

length + 84 bytes for the KB header + 12 bytes for length fields.

Default: 1948
Minimum value: 0
Maximum value (BS2000 systems): 32652, irrespective of the storage medium for the user log file
Maximum value (Unix, Linux and Windows systems): 32668

BS2000 systems:
openUTM uses *length* to determine the block size of the user log file. To do this, openUTM calculates the next largest value of (l*ength* + 100 bytes) that is a multiple of 2 kbytes. openUTM uses this multiple as the block length for the user log file. The 100 bytes comprise of 84 bytes for the KB header + 12 bytes for the record length fields + 4 bytes for the block length fields.

If the user log file USLOG is created on a non-key disk (NK2, NK4), then you must select the value of *length* such that:

length + 100 byte + 16 bytes block-specific internal DVS administration information

is a multiple of 2 Kbytes (on NK2 disks) or 4 Kbytes (on NK4 disks). This allows you to optimally utilize disk space.

The 16 byte block-specific internal DVS administration information are therefore not available for use as user data. You will find more information on this subject in the BS2000 manual "Introductory Guide to DMS".

LSSBS= number

Maximum number of LSSBs (local secondary storage areas) that can be created in a service.

Default: 8
Minimum value: 0
Maximum value: 1600

MOVE-BUNDLE-MSGS= This parameter can be used for an application to allow automatic moving of waiting asynchronous messages from a slave LTERM, slave LPAP or slave OSI-LPAP without a connection to the partner application.

YES When the waiting time defined in MAX CONRTIME has elapsed, or after 10 minutes (if CONRTIME=0), UTM automatically moves FPUT messages to a slave in the bundle with a connection established. It is possible that FPUTs from a transaction will be sent using different slaves from a bundle.

NO

Asynchronous messages to a slave are never sent via another slave.

Default: NO

MP-WAIT=

number

This operand is only supported on BS2000 systems.

Maximum number of seconds for which openUTM waits for a process to sign on to a common memory pool.

Default value: 180
Minimum value: 1
Maximum value: 32000

> **CAUTION!**
> The default value of 180 seconds should only be changed in exceptional circumstances, e.g. if a process terminates with K078 ENQAR and a user dump with the return code KDCSST01.

NB=

length

Maximum length of a working area for

- logical inputs and outputs to and from terminals and transport system applications of the APPLI type
- asynchronous output messages to printers and transport system applications of the SOCKET type

This should be equal to the length of the largest KDCS message area of the program units in bytes.

Default: 2048
Minimum value: 2048
Maximum value (BS2000 systems): 32700
Maximum value (Unix, Linux and Windows systems): 32676

NRCONV= number

(**n**umbe**r** of **conv**ersations)
Maximum number of services that can be simultaneously stacked by the user. NRCONV=0 means that services cannot be stacked.

The following limits are valid:

Number of user IDs + maximum number of services that can be placed on the stack (number of services =*number* * number of user IDs) <= 500000

If the limit value of 500000 is exceeded (by the values specified for NRCONV in the RESERVE statement, see "RESERVE - reserve table locations for UTM objects", and by the number of USER statements, see "USER - define a user ID"), then openUTM automatically creates fewer entries for stacking services. In this case, not all users will be able to place *number* services on the stack.

Default: 0
Minimum value: 0
Maximum value: 15

OSISHMKEY= number

This operand is only supported on Unix, Linux and Windows systems.

Authorization key for the shared memory segment, which is used by OSS for communication based on OSI TP. You must enter a decimal number for *number*. This is a mandatory operand if the application communicates on the basis of OSI TP.

OSI-SCRATCH-AREA= value

Size in KB of an internal UTM working area for dynamic data storage when using the OSI TP protocol.

Default: 256
Minimum value: 128
Maximum value: 32767

On BS2000 systems this working area is automatically extended during runtime, if required.

On Unix, Linux and Windows systems the size of the internal working area must not be modified during runtime. It is recommended that you select the default value. However, if this proves to be insufficient during operation, increase the value of OSI-SCRATCH-AREA and repeat the generation procedure.

PGPOOL= (number,warnlevel1,warnlevel2)

Size of the page pool in UTM pages and the warning levels for utilization of the page pool.

number

Number of UTM pages to be used for the page pool in the KDCFILE (see "Page pool"). The size of each UTM page is defined in the BLKSIZE= operand.

Default: 100
Minimum value: 20
Maximum value: 16777215 - (2 * number of PGPOOLFS)

If you enter a value less than 20, KDCDEF automatically sets PGPOOL=20 without outputting a UTM message.

On Unix, Linux and Windows systems the value of PGPOOL is always an even number. If you enter an uneven number, openUTM subtracts 1 from your entry.

warnlevel1

Numeric value (percentage) indicating the page pool utilization level at which the first warning (UTM message K041) is output.

Default: 80
Minimum value: 1
Maximum value: 99

warnlevel2

Numeric value (percentage) indicating the page pool utilization level at which the second warning is to be output. If *warnlevel2* is exceeded, all asynchronous jobs are rejected. In this case, the user receives a K message, and a corresponding return code is sent to a program unit.

Default: 95
Minimum value: *warnlevel1* + 1
Maximum value: 100

PGPOOLFS= number

Number of files between which the page pool is to be split. If PGPOOLFS = 0, the page pool is located in the main file (on BS2000 systems in the file filebase. KDCA, on Unix, Linux and Windows systems in the file KDCA in the *filebase* file directory). In the case of dual-file operation (MAX ...,KDCFILE=(...,DOUBLE)), the value specified in *number* does not include the two file copies.
The file names are defined by KDCDEF.

Default: 0, i.e. the page pool is located in the main file

Maximum value (BS2000 systems): 99 (and PGPOOL=*number* / 2)
Maximum value (Unix, Linux and Windows systems): 10

Minimum value: The minimum value depends on the number of UTM pages, the UTM page size and the maximum file size permitted on the relevant system.

- On BS2000 systems, an individual UTM file must not be larger than 32 Gbytes in size.
  For BS2000 systems, this results in the following minimum value depending on the size of a UTM page:
  4 if BLKSIZE = 8K and PGPOOL *number* >= 4194304
  2 if BLKSIZE = 4K and PGPOOL *number* >= 8388608
  0: other, for meaning, see above.
- On Unix, Linux and Windows systems in 32-bit mode files up to 2 Gbyte in size are supported.
- On Unix, Linux and Windows systems in 64-bit mode, openUTM can also use larger files as defined by the limits of the operating system and file system.

PGWTTIME= time

Maximum number of seconds for which a program unit can wait for messages to arrive after a blocking call (e.g. PGWT call). During this period, a process of the UTM application is exclusively reserved for this program unit.

Default: corresponds to *time* in TERMWAIT=*time*
Minimum value: 60
Maximum value: 32767

PRINCIPAL-LTH= length

This operand is only supported on BS2000 systems.

Maximum length of a Kerberos principal in bytes. This parameter is only of significance if at least one user is generated with USER ..., PRINCIPAL= or at least one LTERM or TPOOL is generated with KERBEROS-DIALOG=YES. The length of the value specified with USER ... PRINCIPAL= must not be larger than the value generated with MAX PRINCIPAL-LTH=.

When a Kerberos dialog is performed with a client, openUTM saves the Kerberos information in the length resulting from the maximum of this length and the length generated for MAX CARDLTH. If the Kerberos information is longer, it is truncated to this length and stored.
The KDCS call INFO (KCOM=CD) allows the program unit run to read this information if no user signs on to the same client with an ID card after the Kerberos dialog. In this event, the Kerberos information is overwritten by the ID card information.

Default: 0
Minimum value: 0
Maximum value: 100

PRIVILEGED-LTERM= lterm-name

Identifies an LTERM as a privileged connection. Jobs sent to the UTM application via this LTERM are prioritized for processing by UTM in situations in which the UTM application is subject to a high load.

To permit rapid responsiveness even in high-load situations, additional processes (referred to as UTM system processes) are started for a UTM application. The UTM system processes only handle selected jobs. These are primarily internal jobs or jobs issued by an administrator who is signed on at the UTM application via the privileged LTERM. See also operand MAX SYSTEM-TASKS on "MAX - define UTM application parameters".

If optimum use is to be made of this functionality, the PRIVILEGED-LTERM should always be explicitly generated. Only then is it possible for all the mechanisms that allow this LTERM to be privileged in high-load situations to take effect. More specifically, the following approach is recommended:

- The administrator's workstation should be generated via a PTERM- and an LTERM statement.
- The administrator's LTERM should be declared as a PRIVILEGED-LTERM.

If a connection is established via this LTERM then the following applies:

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

The LTERM must be generated as a dialog LTERM in an LTERM statement.

If no PRIVILEGED-LTERM is generated then it is dynamically determined as follows:

- After the start of the application, the first LTERM to which an administrator signs on becomes the privileged LTERM.
- If this administrator then signs off again then the next LTERM to become the privileged LTERM is that at which an administrator signs on or at which an administrator who is already signed on starts a program unit.

QTIME= (qtime1, qtime2)

Specifies the maximum permitted length of time that a service is to wait for the arrival of a message in a message queue. QTIME= refers to user specific (USER queues), permanent (TAC queues) and temporary message queues.

It is possible to define individual maximum values for wait times in dialog or asynchronous services.
If a greater wait time value is specified in a program unit run than is generated in QTIME=, openUTM resets the wait time to the generated value.

qtime1 — Maximum length of wait time for dialog services

qtime2 — Maximum length of wait time for asynchronous services

Both times are specified in seconds.

Default: 32767 (seconds)
Maximum value: 32767 (seconds)
Minimum value: 0 (seconds)

RECBUF= — (number,length)

Size of the transaction-oriented restart area. This area contains the data required for a restart following a transaction or system error. Further information on the restart area can be found on section “Restart area”.

number — Number of UTM pages per process to be used in the KDCFILE to store data for a restart following a system error. The size of each UTM page is defined in the BLKSIZE= operand. If this area is large, the application load is reduced but the restart process following a system error is slower. If this area is small, the application load is increased but the restart process following a system error is faster.

Default: 100 (per process)
Minimum value: 5 (per process)
Maximum value: 32767 (per process)

length — Size in bytes of the buffer available to each application process for temporarily storing restart data. This data is required for a restart following a transaction or system error.

Default: 8192
Minimum value: 1024
Maximum value: 16777212 (16 MB)

RECBUFFS= — number

Number of files between which the restart area is to be split. If RECBUFFS=0, the restart area is located in the main file of KDCFILE. In the case of dual-file operation (MAX ..., KDCFILE=(...,DOUBLE)), the value specified in *number* does not include the two file copies. The file names are defined by KDCDEF.

number must not be greater than the maximum number of processes defined in TASKS=. If this requirement is not fulfilled, the default value is used.

Default: 0
Maximum value (BS2000 systems): 99, or value of the TASKS parameter
Maximum value (Unix, Linux and Windows systems):
10, or value of the TASKS parameter

REDELIVERY= (number1, number2)

Maximum number of redeliveries of an asynchronous message after the service or transaction was rolled back. *number1* and *number2* apply for different message destinations.

number1

Maximum number of redeliveries of messages to an asynchronous TAC. Delivery is always repeated after an asynchronous service was terminated abnormally with PEND ER/FR or system PEND ER without at least one transaction having been completed successfully. Restart of an asynchronous service after PEND RS within the first transaction is not regarded as a redelivery.
When redelivery is made, the program unit assigned to the TAC is restarted. With the FGET call, the number of redeliveries is output in the KB return area.

number2

Maximum number of redeliveries of messages to a service controlled queue. Delivery is always repeated if the message was processed and the transaction was then rolled back.

With the DGET call, the number of redeliveries is output in the KB return area.

Default: (0, 255)
Minimum value for *number1* and *number2*: 0
Maximum value for *number1* and *number2*: 255 (i.e. the number is unlimited)

A value of 0 means that the message is deleted or saved to the dead letter queue after rollback, depending on the value in TAC ...,DEAD-LETTER-Q.

If the value is set to 255, a message is redelivered any number of times. Note that this can result in an endless loop if, for example, a program unit is rolled back because of a programming error. Additionally the message cannot be saved to the dead letter queue in case of an endless loop.

REQNR= number

This operand is only supported on BS2000 systems.

Maximum number of PAM read/write jobs that can be issued in parallel at the same time for a file in a UTM process. This value can be used to control the parallel processing of input/output operations within certain limits.

Default: 20
Minimum value: 1
Maximum value: 100
KDCDEF replaces an invalid value with the maximum value without outputting a message.

RESWAIT= (time1,time2)

(**res**ource **wait**)
The times specified for *time1* and *time2* can be modified during runtime using the administration command KDCAPPL.

time1

Maximum number of seconds for which a program unit can wait for a resource locked by another transaction: GSSBs, TLSs, ULSs, and on BS2000 systems possibly LTERM partners if ANNOAMSG=N.
If the resource does not become available within this time, the program unit receives an appropriate return code.
If the transaction currently occupying the resource is waiting for an input message following a PEND KP or PGWT KP program call, the program unit receives an appropriate return code immediately without having to wait for the period specified in *time1*. If a PEND KP or PGWT KP call is issued in a blocking transaction, all pending program units are informed of this by means of a return code.

RESWAIT=0: The application program does not wait for the resource to become available. If the resource is locked by another transaction, the requesting program unit immediately receives an appropriate return code.

Default: 120
Minimum value: 0
Maximum value: 32767

> **i** On BS2000 systems the real waiting time depends on the precision with which the bourse waiting time was set in the operating system.

time2

Maximum number of seconds for which you can wait for a resource locked by another process. If *time2* is exceeded, the application is terminated abnormally.

time2 should not be set too low, since certain activities in the UTM application must be performed and completed by a process before the same activities can be initiated in another process.

Example

When sending a message, a process locks the terminal to which the message is directed. If another process wishes to access an input message of the same terminal, it must wait for the terminal to become available again.

In particular, the value entered for *time2* must be at least equal to the longest processing time (real time) required in the following cases:

- In the case of a communication partner generated with PTERM ..., PTYPE=APPLI, the resources are locked for the entire duration of a processing step. This includes the time required to process the event exit VORGANG at the start and/or end of a conversation.
- At the end of a conversation, the resources remain locked as long as the event exit VORGANG is running.

Default: 300
Minimum value: 300
Maximum value: 32767

If the value 0 is specified for *time2*, KDCDEF uses the default value 300 without outputting a UTM message. If you specify a value between 0 and 300, however, KDCDEF issues an appropriate UTM message.

SAT=

(**s**ecurity **a**udit **t**rail)

This operand is only supported on BS2000 systems.

Minimum event logging with SAT. Further information about "SAT logging" can be found in the openUTM manual “Using UTM Applications on BS2000 Systems”.

ON

SAT logging is switched on.

Minimum logging with SAT is switched on for the following events:

- signing a process on to and off from the UTM application
- switching the memory protection key
- exchanging programs
- executing a UTM SAT administration command.

Minimum logging can be extended and controlled by means of preselection. This is generated using the SATSEL statement and the SATSEL= operand in the USER and TAC statements. The administration command KDCMSAT can be used to modify the preselection values defined during generation.

OFF

SAT logging is switched off
The logging procedure only covers attempts to access the SAT administration TAC KDCMSAT (apart from KDCMSAT HELP). All other events are ignored. SAT logging can be switched on and off using the SAT administration TAC KDCMSAT (see the openUTM manual “Using UTM Applications on BS2000 Systems”).

Default: OFF

SEMARRAY=

(number,number1)

This operand is only supported on Unix, Linux and Windows systems.

Range of semaphore keys for global semaphores (process synchronization). Semaphore keys are global parameters under the Unix, Linux and Windows systems. With SEMARRAY, you enter an initial value *number* and an upper limit *number*. openUTM then reserves these keys, incrementing them by 1 starting with the initial value. For further information, please contact your system administrator.

This is a mandatory operand if SEMKEY= is not specified.

> The SEMARRAY= and SEMKEY= parameters are mutually exclusive. Compared to SEMKEY=, SEMARRAY= offers the advantage of allowing openUTM to reserve more than ten semaphore keys. To calculate the number of semaphore keys required for a UTM application, please refer to the description of the global system resources in the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

number

Initial value (numeric value)

number1

Number of keys to be reserved

Minimum value: 1
Maximum value: 1000

SEMKEY= (number,...)

This operand is only supported on Unix, Linux and Windows systems.
Semaphore keys (**sem**aphore **key**) for global semaphores (process synchronization).
Semaphore keys are global parameters under the Unix, Linux and Windows systems. You can define up to 10 semaphore keys in a list. All semaphore keys (*number,...*) are specified in the form of a decimal number. For further information, please contact your system administrator.

This is a mandatory operand if SEMARRAY is not specified.

> The SEMARRAY= and SEMKEY= parameters are mutually exclusive. Compared to SEMKEY=, SEMARRAY= offers the advantage of allowing openUTM to reserve more than ten semaphore keys. To calculate the number of semaphore keys required for a UTM application, please refer to the description of the global system resources in the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

SM2= This defines whether the UTM application is to supply data to SM2 or openSM2 for performance monitoring.

NO Performance monitoring with openSM2 is generally prohibited for the UTM application, i.e. the UTM application cannot supply data to openSM2, nor can this be explicitly activated by the UTM administrator.

OFF The UTM application can supply data to openSM2, but this must be explicitly activated by the administrator using KDCAPPL SM2=ON. The supply of data can be deactivated again at any time using the administration command KDCAPPL SM2=OFF.

Default value: OFF

ON The UTM application can supply data to openSM2. This is activated automatically when starting the UTM application. It can be deactivated again at any time by the UTM application administrator using the administration command KDCAPPL SM2=OFF.

SPAB= length

Maximum length of the standard primary working area in bytes

Default: 512
Minimum value: 0
Maximum value: 32767

STATISTICS-MSG= Specifies whether or not openUTM is to produce statistics message K081 hourly.

FULL-HOUR

Statistics message K081 is produced every hour and written in the SYSLOG. At the same time, openUTM resets the following application specific statistical values to 0:

- number of messages received (*term_input_msgs*)
- number of messages sent/output (*term_output_msgs*)
- number of requests to write records in the user log file USLOG (*logfile_writes*)
- percentage of requests from buffers in the cache that led to wait times (*cache_wait_buffer*)

NONE

Statistics message K081 is not produced and the statistical values listed above are not automatically reset to 0.
You should choose NONE if you want to reset the statistical values listed above via the administration when needed (see the openUTM manual “Administering Applications”, KC_MODIFY_OBJECT).

Default: FULL-HOUR

SYSLOG-SIZE= size

Automatic size monitoring of the system log file SYSLOG by openUTM.

- *size* !=0
  This can only be specified if the system log file SYSLOG is created as a file generation group (FGG). If SYSLOG is a normal file and a value other than 0 is entered for *size*, openUTM aborts the application startup with the start error 58. If SYSLOG is created as an FGG, you can use SYSLOG-SIZE to activate the automatic size monitoring of the SYSLOG by openUTM. In this case, *size* defines the file generation size at which openUTM switches to the next file generation.
- *size*=0
  If the value 0 is specified for *size* (default), openUTM does not monitor the size of the SYSLOG file. Instead, it outputs all UTM messages directed to SYSLOG to the same file generation until openUTM switches to another file generation by means of administration (KDCSLOG command), or until size monitoring is activated.
- *size*>=100
  Values >= 100 are interpreted by openUTM as follows: the size of each individual SYSLOG file generation must not exceed the value (*size* * size of a UTM page). The size of each UTM page is defined in BLKSIZE. When the size of the SYSLOG file exceeds this threshold value, openUTM automatically switches to the next SYSLOG file generation.
- *size*<100
  openUTM automatically resets values between 1 and 99 to 100. In this case, a UTM message is output for information purposes.
- *size*<0
  Values < 0 are rejected by KDCDEF.

The administrator can modify the generated threshold value, and activate or deactivate size monitoring as desired during operation (e.g. with the KDCSLOG command).

Default: 0 (no size monitoring)
Minimum value: 100
Maximum value: $(2^{31} - 1)$

SYSTEM-TASKS

Controls the number of UTM system processes.
Under load, in the case of UTM applications all processes can be utilized by program unit runs, and they are then not available for processing other jobs.

To ensure that an application continues to be responsive and, for example, can also process internal jobs to terminate transactions, communication between the nodes of a UTM cluster application or an administrator's jobs in these situations, UTM starts further processes, known as UTM system processes, in addition to the processes generated and started by the user.

Generally no program unit runs are executed by the UTM system processes, and they are only used for internal jobs in bottleneck situations. Consequently the additional UTM system processes only place a slight load on the host.
The UTM system processes are started independently by UTM and in addition to the processes started by the user.
See also the MAX operand PRIVILEGED-LTERM in section "MAX - define UTM application parameters".

*STD

*STD means that UTM starts (up to) three additional processes for the application; these are then used as UTM system processes. Depending on the number of tasks started for the application, the second, fourth, and seventh processes of an application become UTM system processes.

*STD is the default value.

number

Maximum number of UTM system processes which are to be started in addition for the application.

The value 0 means that no UTM system process will be started.

Minimum value: 0
Maximum value: 10

Values greater than 10 are ignored and rounded down to 10.

The table below shows how many UTM system processes are started additionally for the generated value SYSTEM-TASKS=*STD in accordance with the start parameter TASKS:

Start parameter TASKS=	Number of additionally started UTM system processes	Total started processes
1	0	1
2	1	3
3	2	5
4	2	6
5	3	8
n > 5	3	n + 3

If more than three UTM system processes are generated, depending on the value of SYSTEM-TASKS and the number of started processes, the 11th, 21st, 31st, 41st, 51st, 61st, and 71st processes also become UTM system processes.

TASKS=

number

Maximum number of processes that can be used simultaneously for the application.

This is a mandatory operand.

Minimum value: 2
Maximum value: 240

KDCDEF automatically resets values < 2 to 2 without outputting a UTM message.

The current number of processes is defined when starting the application. You can specify TASKS=1 during startup. The administrator can dynamically modify the number of processes during runtime (e.g. with the administration command KDCAPPL). The number of processes specified during startup or set by the administrator must not exceed the value generated here.

TASKS-IN-PGWT= number

Maximum number of processes of the UTM application in which program units with blocking calls, e.g. the KDCS call PGWT, may run simultaneously. The value of TASKS-IN-PGWT must be less than that of the TASKS= operand.

If TASKS-IN-PGWT=0, it is not possible to generate a TAC class or a transaction code (TAC) for which blocking calls are permitted (see TAC/TACCLASS ..., PGWT=). In this case, PGWT=NO must be specified in all TACCLASS and TAC statement (see also the TAC statement in section"TAC - define the properties of transaction codes and TAC queues" and the TACCLASS statement in section "TACCLASS - define the number of processes for a TAC class" for more information).

Default value: 0
Minimum value: 0
Maximum value: *number* in TASKS -1

TERMWAIT= time

(**term**inal **wait**)
Maximum time in seconds that may elapse in a multi-step transaction (i.e. after PEND KP) between dialog output to the partner and the subsequent dialog response from the partner. This value applies for all dialogs in which the partner assumes the client role (terminals, UPIC clients, OSI TP, LU6.1 and LU6.2 job submitters). For terminal clients, for example, *time* is the time the user has to think after PEND KP. In the event of a timeout, the transaction is rolled back and the resources reserved by the transaction are released. The connection to the partner is shut down.

Default: 600
Maximum value: 32767
Minimum value: 60

TRACEREC= number

Maximum number of entries in the process-specific trace areas handled by openUTM. This value applies to the trace area

- of the main routine KDCROOT (UTM Diagarea)
- of the UTM system code (KTA trace)
- of the XAPTP module (XAP trace) for OSI TP applications

openUTM writes trace information to these areas for diagnostic purposes.

Length of the entries:

- Entry in UTM Diagarea: 138 bytes (on 32-bit systems) or 256 bytes (on 64-bit systems)
- KTA and XAP trace entry: 64 bytes (on 32-bit systems) or 112 bytes (on 64-bit systems

Default: 32500
Minimum value: 1
Maximum value: 32500 (depending on the available resources)

KDCDEF automatically resets values < 1 to the default value and values > 32500 to the maximum value without outputting a UTM message.

TRMSGLTH= length

This defines the maximum value for the following:

- The length of physical output messages sent to a terminal, printer or transport system application (PTYPE=APPLI) or received by a terminal or transport system application with PTYPE=APPLI. When the message length is calculated, all characters to be transmitted, including control characters etc., must be included.
- The length of asynchronous output messages to transport system applications of the SOCKET type.
- The length of the message section of the input message received from an UPIC client that uses TCP/IP via the socket interface. During the calculation of the length, it is necessary to take account of all the characters that are to be transferred, including protocol elements.

Default: 32700 bytes

Maximum value: 32700 bytes
A value < 32700 is replaced with the maximum value by KDCDEF with no message. Values < 32700 are only supported for compatibility reasons.

If you use RSO printers, the size of the RSO buffer (REMOTE-BUFFER- SIZE in the SPOOL parameter file) must be greater than or equal to 32700. See also section "Defining the RSO buffer size" (Entries for RSO and SPOOL) for more information.

USLOG= This defines single- or dual-file operation for the user log file USLOG.

SINGLE	Single-file operation is activated for the user log file. Default: SINGLE
DOUBLE	For security reasons, dual-file operation is activated for the user log file. Further information on the user log file can be found in openUTM manual “Using UTM Applications”.
VGMSIZE=	number This operand is only supported on BS2000 systems. This parameter is used to generate a buffer area with the specified size for the service memory of an SQL database system. It also restricts the user’s share of the page pool. VGMSIZE= is specified in KB. If the service memory area to be logged when the PEND call is issued is greater than *number*, the service is terminated with PEND ER. Default value: 32KB Minimum value: 32KB Maximum value: 256KB
XAPTPSHMKEY=	number This operand is only supported on Unix, Linux and Windows systems. Authorization key for the XAPTP shared memory segment Shared memory keys are global system parameters. XAPTPSHMKEY is a mandatory operand if the application is to communicate via the OSI TP protocol.

The table below provides an overview of the purpose and default values of the individual operands of the MAX statement:

Operand	Purpose	Mandatory	Default value
Operands valid for all operating systems			
APPLIMODE=	Choice of UTM variant: UTM-S or UTM-F		SECURE
APPLINAME=	Name of the UTM application	X	-
ASYNTASKS=	Asynchronous processing (number of processes for asynchronous processing and asynchronous services open at the same time)		1, 1
BLKSIZE=	Size of a UTM page		2K (in UTM cluster applications: 4K or 8K)
CACHESIZE=	Tuning feature (size and properties of the cache)		Depending on the system: BS2000 systems: (1024,70%, NORES, PS) Unix, Linux and Windows systems (1024,70%)
CLRCH=	Character for overwriting the communication area and standard primary working area		None
CONN-USERS=	Restriction on the number of users or clients active simultaneously		Depending on the system: BS2000 systems: No restriction Unix, Linux and Windows systems: Mandatory operand

Operand	Purpose	Mandatory	Default value
CONRTIME=	Automatic connection setup for printers (waiting time for reconnection)		10 minutes
DATA-COMPRESSION	Controlling data compression		STD
DEAD-LETTER-Q-ALARM=	Monitors the number of messages received in the dead letter queue		0, i.e. no monitoring
DESTADM=	Asynchronous administration		None
DPUTLIMIT1=	Time-driven jobs (upper limit)		360 days
DPUTLIMIT2=	Time-driven jobs (lower limit)		1 day
GSSBS=	GSSB storage areas (maximum number)		32
HOSTNAME=	Virtual host name for the UTM application		8 blanks
KB=	Maximum length of the communication area		512
KDCFILE=	Assigning a KDCFILE	X	-
KEYVALUE=	Data access control using the lock/key code concept (number of the highest key code)		32
LEADING-SPACES=	Pass leading blanks in messages from terminals or from TS applications (PTYPE=APPLI/SOCKET) to the program unit		NO
LPUTBUF=	Logging of user data with LPUT (number of PAM pages in the page pool)		1
LPUTLTH=	Logging of user data with LPUT (maximum LPUT message length)		1948 bytes
LSSBS=	LSSB storage areas (maximum number)		8
MOVE-BUNDLE-MSGS=NO	Automatic moving of waiting asynchronous messages of a slave LTERM, a slave LPAP, or a slave OSI-LPAP		NO
NB=	Maximum length of the KDCS message area		2048
NRCONV=	Maximum number of stacked services		0

Operand	Purpose	Mandatory	Default value
OSI-SCRATCH-AREA=	Size in KB of an internal UTM working area		256
PGPOOL=	Size of the page pool and warning levels		100 UTM pages, 80%, 95%
PGPOOLFS=	Tuning feature: splitting the page pool		Page pool in KDCFILE
PGWTTIME=	Maximum time for the KDCS call PGWT		TERMWAIT= *time*
PRIVILEGED-LTERM=	Define the privileged LTERM		-
QTIME	Maximum permitted wait time for messages from service controlled queues		32767 seconds
RECBUF=	Tuning feature: size of the restart area in KDCFILE or process-oriented system memory		5 PAM pages per process, 512 bytes
RECBUFFS=	Tuning feature: splitting the restart area		in KDCFILE
REDELIVERY=	Maximum number of redeliveries of an asynchronous message		0 for UTM-controlled queues, 255 for servicecontrolled queues
RESWAIT=	Waiting time for a resource (e.g. GSSB, TLS) locked by another transaction (time1) or process (time2)		120 seconds, 300 seconds
SPAB=	Maximum SPAB length		512
SM2=	Permitting, activating, and deactivating the supply of UTM data to SM2		OFF
STATISTICS-MSG=	Statistics message K081 is produced and the counter is automatically reset to 0		FULL-HOUR
SYSLOG-SIZE=	Automatic size monitoring of the SYSLOG file by openUTM		0
SYSTEM-TASKS	Number of UTM system processes		*STD

Operand	Purpose	Mandatory	Default value
TASKS=	Number of UTM processes	X	-
TASKS-IN-PGWT=	Number of processes for PGWT jobs		0
TERMWAIT=	Maximum waiting time for dialog input within a transaction		600 seconds
TRACEREC=	Space reserved for diagnostic information (number of entries)		32500
TRMSGLTH=	Maximum message length		32700 bytes
USLOG=	Single- or dual-file operation of the user log file		SINGLE
VGMSIZE=	Generate the buffer area with the specified size		32 KB

Operand	Purpose	Mandatory	Default value
BS2000-specific operands			
BRETRYNR=	Communication with BCAM (number of retries when sending messages)		10
CARDLTH=	ID card reader for KDCSIGN check		0
CATID=	Catalog IDs for the KDCFILE		Default CATID
LOCALE=	Default language environment		Blanks
LOGACKWAIT=	Support for output devices (waiting time for confirmation)		600 seconds
MP-WAIT=	Maximum waiting time per process for connection to the common memory pool		180 seconds
PRINCIPAL-LTH=	Maximum length of a Kerberos principal in bytes		32
REQNR=	Tuning feature: PAM I/O jobs (maximum number of parallel jobs)		20
SAT=	Minimum logging of events with SAT		OFF
VGMSIZE=	Size of the buffer area for the service memory of an SQL database system		32KB

Operand	Purpose	Mandatory	Default value
Unix, Linux and Windows system-specific operands			
CACHESHMKEY=	Authorization key for a shared memory segment (global buffer for file access)	X	-
IPCSHMKEY=	Authorization key for a shared memory segment (communication between UTM processes)	X	-
IPCTRACE=	Number of UTM entries in the IPC trace area		1060
KAASHMKEY=	Authorization key for a shared memory segment (global data)	X	-
OSISHMKEY=	Authorization key for an OSS shared memory segment	with OSI TP	-
SEMARRAY=	Range of semaphore keys for global semaphores (alternative to SEMKEY)	X	-
SEMKEY=	Semaphore keys for global semaphores (alternative to SEMARRAY)	X	-
XAPTPSHMKEY=	Authorization key for the XAPTP shared memory segment	with OSI TP	-

6.5.29 MESSAGE - define a UTM message module

The MESSAGE control statement allows you to incorporate user message modules in the configuration. It is possible to use a separate user message module to adapt the message texts and/or the message destinations of individual messages to suit your requirements.

For more information on message modules see also section "UTM messages" in this manual and the openUTM manual ”Messages, Debugging and Diagnostics”.

Generating message modules on BS2000 systems

In order to internationalize the application, it is possible to create several user message modules which output the UTM messages of an application in the appropriate language.

The respective language environment can be defined for a user message module by means of a locale, i.e. a unique pair of language and territorial identifiers. The language-specific message modules are assigned for message output in accordance with the locale defined for the user and LTERM partner.

The German UTM message module KCSMSGS and the standard English UTM message module KCSMSGSE are supplied with openUTM.

`MESSAGE`	`MODULE=name`
	`[ ,LIB=omlname ]`
	`[ ,LOCALE = (lang-id [,terr-id]) ]`

MODULE= name

Name of the user-specific message module up to eight characters in length. This module is created using the KDCMMOD tool (see the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”).

This is a mandatory operand.

The name specified here must be unique within the application.

LIB= omlname

Object module library from which the user-specific message module is to be loaded dynamically. *omlname* can be up to 54 characters in length.

If the user-specific message module is to be loaded dynamically, it must not be linked to the application.

If nothing is specified for LIB= , TASKLIB is assumed. This does **not** correspond to the SET-TASKLIB command, rather a library named TASKLIB must exist in this case. Dynamic loading of the user message module from the library assigned with SYSFILE-TASKLIB is not supported.

> When loading dynamically, the DBL searches for the user message module first in the library that you have assigned in LIB= . If this library does not exist, the DBL aborts the search. If the library exists but the user message module could not be found there, the DBL searches through the alternative libraries. The alternative libraries are those that have been assigned a file link name BLSLIB*nn* (0<=*nn*<=99).

LOCALE= (lang_id, terr_id)

Language environment of the user-specific message modules defined by means of a language identifier and possibly a territorial identifier. By making the appropriate entries in the LOCALE= parameter of the USER or LTERM statement, you can assign a corresponding UTM message module. Messages are then output in the user's language.

If you issue more than one MESSAGE statement, each statement must contain the LOCALE= parameter. The *lang_id* and *terr_id* combination must be unique in each MESSAGE statement for a UTM message module.

lang_id Freely selectable language identifier for a UTM message module, up to two characters in length.

There is no default value for *lang_id*, i.e. this is a mandatory parameter.

terr_id Territorial identifier for a UTM message module up to two characters in length. You can also specify blanks for *terr_id*.

If you specify MESSAGE ...,LOCALE=, you must also define the MAX ...,LOCALE (see "MAX - define UTM application parameters"). The application message module of the UTM application is automatically the message module whose *lang_id* and *terr_id* in the MESSAGE statement match the locale in the MAX statement.
openUTM uses the application message module for messages to SYSLST, SYSOUT and CONSOLE. The message destinations specified in the other message modules have no significance.

The UTM message module whose *lang_id* and *terr_id* in the MESSAGE statement are identical to the values entered for LOCALE= in the USER or LTERM statement is used for messages to STATION, SYSLINE and PARTNER.

Specifications relating to the user have priority over those relating to the LTERM partner, i.e. if a user is signed on when a message is output, openUTM uses the UTM message module appropriate for that user. If the UTM message modules are assigned using language and territorial identifiers, the procedure is as follows:

- If a UTM message module exists with a *lang_id* and *terr_id* combination identical to the entries in the USER or LTERM statement, UTM messages are output in this language environment.
- If an identical combination cannot be found, the UTM message module with the same *lang_id* but for which no *terr_id* has been generated is used.
- If this is not possible, the application message module is used.

Generating message modules on Unix, Linux and Windows systems

On UnixOn Unix, Linux and Windows systems, you can generate exactly one user-defined message module with the MESSAGE statement, i.e. you may only specify the MESSAGE statement once within a single KDCDEF run.

If a MESSAGE statement is not issued, the name of the external C/C++ structure is KCSMSGS. An object module with a C/C++ structure with this name is supplied with openUTM as a file.

On Unix and Linux systems, the file is the object module `kcsmsgs.o` in the library `libwork` under the path *utmpath* `/sys`.

On Windows systems the module `kcsmsgs.obj` in the library *utmpath* `/sys/libwork.lib`.

MESSAGE	MODULE=name

MODULE= name

Name of the external C/C++ structure with which messages are addressed. In the case of a user-specific message module (see the description of the KDCMMOD tool in the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”), the name specified here must match the name of this module. *name* can be up to eight characters in length.

This is a mandatory operand.

6.5.30 MPOOL - define a common memory pool (BS2000 systems)

The MPOOL control statement allows you to define the properties of a common memory pools.

The MPOOL statement can be issued several times, the only limit being the number of pools that can be created by a single process. Support is provided for up to eight common memory pools with SCOPE=GROUP or SCOPE=GLOBAL under a single user ID.

The common memory pools are always created as FIXED. Every task that connects to an existing common memory pool is assigned the same address as the task that set up the common memory pool.

The sequence of MPOOL statements within the generation run determines the order in which the common memory pools are created. Firstly, all common memory pools generated with SCOPE=GLOBAL are created in accordance with the sequence of MPOOL statements. This is followed by the creation of all common memory pools generated with SCOPE=GROUP, as defined by the sequence of MPOOL statements.

```
MPOOL  poolname
       [ ,ACCESS={ READ | WRITE } ]
       [ ,PAGE=X'xxxxxxxx' ]
       [ ,SCOPE={ GROUP | GLOBAL } ]
         ,SIZE=poolsize
```

poolname
: Name of the common memory pool. *poolname* must be unique within the UTM application and can be up to 50 characters in length.

  A number is appended to the name.

ACCESS=
: Access authorization

  READ
  : Read-only access to the common memory pool

    Default: READ

  WRITE
  : Read and write access to the common memory pool

PAGE= X’xxxxxxxx’

Hexadecimal address in the format X’xxxxxxxx’.

- 24-bit addressing mode: If the address is not a multiple of 64K (the four low-order half-bytes are 0), it is rounded off to a multiple of 64K.
- 31-bit addressing mode: The address is a multiple of 1MB. If this is not the case, it is rounded off to a multiple of 1MB.

Default:

- 24-bit addressing mode: The pool is created starting with the lowest possible address.
- 31-bit addressing mode: The pool is created starting with the lowest possible address above X'01000000'.

> **i** If, on BS2000, global common memory pools are used in several UTM applications with the same contents/names, the parameter PAGE=X'xxxxxxxx' must be specified with the same address in all applications. The address specified using PAGE= must be selected in such a way that the address area reserved is available in all these applications.

The common memory pools are always created as FIXED, i.e. all tasks of the UTM application find the pool at the same address in their virtual address space.

An alternative to the use of PAGE= is to ensure that all the shared pools are generated in the same sequence in all applications. The MPOOL statements for shared pools must be specified at the beginning of the MPOOL statements.

SCOPE= Scope of the memory pool

GLOBAL All processes in the system

GROUP All processes that run under the same user ID.

Default: GROUP

SIZE= poolsize

Number of 64 KB memory segments in the pool (1 unit corresponds to 64KB)

In 31-bit addressing mode, the memory segments are 1MB in length. The size of the common memory pool is thus rounded up to the nearest MB, which is calculated by multiplying *poolsize* by 64KB.

This is a mandatory operand.

6.5.31 MSG-DEST - define user-specific messages destinations

This statement allows you to define up to four additional user-specific message destinations for the UTM messages.

For this purpose, openUTM provides the unoccupied UTM message destinations, USER-DEST-1, USER-DEST-2, USER-DEST-3 and USER-DEST-4. MSG-DEST allows you to assign these UTM message destinations to concrete destinations. These destinations may be:

- a USER queue, or in other words, the message queue of a user ID
- a TAC queue
- an asynchronous TAC
- or an LTERM partner, that is not assigned to a UPIC client.

You can also assign several message destinations of the same type, for example, three LTERM partners and one USER queue. By defining the USER or TAC queue as the user specific message destination you can ensure that the UTM messages are output to the WinAdmin or WebAdmin administration workstation. More information can be found in the openUTM manual ”Messages, Debugging and Diagnostics” as well as in the online help of WinAdmin and WebAdmin, keyword „message collector“.

`MSG-DEST`	`msgdest`
	`,NAME=name`
	`,DEST-TYPE={ LTERM | USER-QUEUE | TAC }`
	`[ ,MSG-FORMAT={ FILE | PRINT } ]`

msgdest

Name of the UTM message destination to which you wish to assign a user specific message destination. Possible values are:

USER-DEST-1, USER-DEST-2, USER-DEST-3 or USER-DEST-4.

msgdest must also be assigned, using KDCMMOD, to the messages you wish to output to this user-specific message destination. For more information see section "User-specific message destinations" and the description of KDCMMOD in the openUTM manual ” Messages, Debugging and Diagnostics”.

NAME= name

Name of the user-specific message destination. Possible values are:

- Name of a UTM user ID. This must be generated in a USER statement.
- Name of an asynchronous TAC. This must be generated in a TAC statement with TYPE=A.
- Name of a TAC queue. This must be generated in a TAC statement with TYPE=Q.
- *BS2000 systems:*
  Name of an LTERM partner. This must be generated in an LTERM application and may not be assigned to a PTERM with PTYPE=UPIC-R.
- *Unix, Linux and Windows systems:*
  Name of an LTERM partner. This must be generated in an LTERM application and may not be assigned to a PTERM with PTYPE=UPIC-R or UPIC-L.

All messages that are linked via KDCMMOD to *msgdest* are then also output to the destination specified in *name*.

> **i** User-specific message destinations should not be locked or dynamically deleted because otherwise no more messages will be output at this destination.

DEST-TYPE= Specifies the type of the message destination in *name*:

LTERM The message destination specified in *name* is an LTERM partner.

TAC The message destination specified in *name* is an asynchronous TAC or a TAC queue.

USER-QUEUE The message destination specified in *name* is a USER queue.

MSG-FORMAT= Specifies the format in which the message is passed to the message destination.

FILE The format corresponds to the data structures for the MSGTAC program. So only message inserts without messages texts are passed, the message inserts are not converted to a printable format.

PRINT The format corresponds to the output format of the UTM tool KDCPSYSL. So the message is prefixed with the date and time, followed by the message text with the text inserts and additional inserts. All inserts are formatted printable.

KDCPSYSL is described in the openUTM manual “Using UTM Applications”.

Default: FILE

6.5.32 MUX - define a multiplex connection (BS2000 systems)

The MUX control statement allows you to define the name and properties of a multiplex connection between the UTM application and a Session Manager (OMNIS). This multiplex connection can then be used simultaneously by several terminals to sign on to the UTM application.

The initiative for establishing the transport connection between openUTM and the Session Manager can come from either side, but only the Session Manager can open a session.

```
MUX   name
      [ ,BCAMAPPL=local_appliname ]
      [ ,CONNECT={ Y | N } ]
      [ ,MAXSES=number ]
      [ ,NETPRIO={ M EDIUM | LOW } ]
        ,PRONAM={ processorname | C'processorname' }
      [ ,STATUS={ ON | OFF} ]
```

name — Name of the multiplex connection

> The specified name must be unique and must not be assigned to any other object in name class 3. See also section “Uniqueness of names and addresses"

BCAMAPPL= local_appliname

Local name of the UTM application as defined in the MAX statement (APPLINAME on "MAX - define UTM application parameters") or BCAMAPPL statement (see "BCAMAPPL - define additional application names"). This name is then used to establish a connection to the Session Manager, i.e. the Session Manager must specify *local_appliname* as the partner name when connecting to the UTM application. By issuing several MUX statements with different BCAMAPPL names, you can set up parallel connections to the Session Manager.

Default:
Application name defined in the statement MAX APPLINAME=*appliname*

CONNECT= Set up the local transport connection on application start

Y — When starting the application, openUTM attempts to establish a logical transport connection to the Session Manager.

If unsuccessful, openUTM repeats its attempt to establish the connection at intervals defined in MAX ...,CONRTIME=*time*.

Default: Y

N — When starting the application, openUTM does not attempt to establish a connection to the Session Manager.

MAXSES= number

Maximum number of simultaneously active sessions between the Session Manager and the UTM application

> openUTM creates *number* LTERM partners internally for the specified number of sessions. The number of LTERM partners must be taken into consideration in the maximum number of UTM names. See section “Maximum values for names” (Number of names).
>
> Default value: 10
> Minimum value: 1
> Maximum value: 65000 (theoretical value)

NETPRIO= Transport priority to be used on the transport connection between the Session Manager and the UTM application

Default: MEDIUM

PRONAM= { processorname | C’processorname’ }

Name of the system on which the Session Manager is located.
If the *processorname* contains special characters it must be entered as a character string using C’...’.

STATUS= Status of the multiplex connection

ON The connection to the Session Manager is not locked.

Default: ON

OFF The connection to the Session Manager is locked. A connection cannot be established between the Session Manager and the UTM application.

This status can be modified by the administrator.

6.5.33 OPTION - manage the KDCDEF run

The OPTION control statement allows you to manage the KDCDEF run.

The control statements for KDCDEF can be distributed in such a manner that only the OPTION statements are contained in a procedure file/shell script, but the actual generation statements are read from other files (on BS2000 systems from SAM or ISAM files or from LMS library elements).

openUTM only processes OPTION statements if they have been read in by SYSDTA or *stdin*.

OPTION statements are ignored by KDCDEF, if is read from a file that is assigned using OPTION DATA=.

If you issue more than one OPTION statement, the values last specified are taken as valid.

If the OPTION statement is not specified, only the KDCFILE is generated, i.e. the default setting GEN=KDCFILE applies.

```
OPTION  [ ,DATA= { filename |
                   *LIBRARY-ELEMENT [1] (LIBRARY=lib-name
                                        ,ELEMENT=element
                                       [ ,VERSION=C'version' |
                                                  *HIGH EST-EXISTING |
                                                  *UPPER-LIMIT ]
                                       [ ,TYPE=element-type ]) } ]
        [ ,GEN= { KDCFILE | ROOTSRC | NO | ALL |  CLUSTER[2] |

                  (KDCFILE,ROOTSRC)          |
                  (CLUSTER,KDCFILE)[2]       |
                  (CLUSTER,ROOTSRC)[2]       |
                  (CLUSTER,KDCFILE,ROOTSRC)[2] } ]
        [ ,GEN-RSA-KEYS={ YES | NO } ]

        additional opernan on BS2000 systems
        [ ,ROOTSRC=filename ]

         additional operand on Unix, Linux and Windows systems
        [ ,CHECK-RFC1006={ NO | YES } ]
```

[1] only on BS2000 systems
[2] only on Unix, Linux and Windows systems

CHECK-RFC1006=

This parameter is only supported on Unix, Linux and Windows systems.

Extended check of the UTM generation for the communication via TCP/IP connections with RFC1006.

YES	KDCDEF checks the specifications of transport addresses for all communication partners and local transport system end points that are generated with T-PROT= RFC1006 for completeness and plausibility. When OPTION CHECK-RFC1006=YES, a port number must be specified in the LISTENER-PORT parameters of the ACCESS-POINT, BCAMAPPL, CON, OSI-CON, and PTERM statements. Default: YES
NO	KDCDEF does not execute any extended checks.
DATA=	Specifies the source from which the subsequent KDCDEF control statements are to be read. The source can also have been generated by inverse KDCDEF by means of the statement CREATE-CONTROL-STATEMENTS. For information on the inverse KDCDEF function, see section "Inverse KDCDEF".
filename	The KDCDEF control statements are read from the file specified here (on BS2000 systems, from a SAM or ISAM file). If the end-of-file is reached then the next KDCDEF control statements are read from SYSDTA or *stdin* again.
*LIBRARY-ELEMENT(...)	This parameter value is only supported on BS2000 systems. The KDCDEF control statements are read from the LMS library element specified here. If the end of the file is reached then the next KDCDEF control statements are again read from SYSDTA. If the specified library element does not exist then KDCDEF cancels the generation run with an error message
LIBRARY=	lib-name Name of an LMS library. The file name can be up to 54 characters in length. LIBRARY is a mandatory parameter.
ELEMENT=	element Name of an LMS element. The element name may be up to 64 characters in length and consists of an alphanumeric string which can be subdivided into multiple substrings separated by periods or hyphens. ELEMENT is a mandatory parameter.
VERSION =	Version of the LMS element.
C'version'	The element version is specified as an alphanumeric string of up to 24 characters in length which can be subdivided into multiple substrings separated by periods or hyphens.
*HIGHEST-EXISTING	The highest version of the specified element present in the library is read. Default: *HIGHEST-EXISTING

*UPPER-LIMIT	The highest possible version of the specified element is read. LMS indicates this version by means of an "@".
TYPE=	element-type Type of LMS element. An alphanumeric string of up to 8 characters in length can be specified for the type. Default value: S i For further information on the syntax rules for the names of LMS elements and a specification of version and type, see the manual "LMS SDF Format".
GEN=	Specifies what objects are to be generated.
KDCFILE	The KDCFILE is generated. Default: KDCFILE
ROOTSRC	The ROOT table source is generated.
(KDCFILE,ROOTSRC)	The KDCFILE and the ROOT table source are generated.
CLUSTER	This parameter value is only supported on Unix, Linux and Windows systems. The following UTM cluster files are generated: • the cluster configuration file • the cluster user file • the cluster page pool files • the cluster GSSB file • the cluster ULS file These files must not already exist. i If you have specified OPTION GEN=CLUSTER or (CLUSTER,...), you must also specify a CLUSTER statement and at least two CLUSTER-NODE statements.
(CLUSTER,KDCFILE)	This parameter value is only supported on Unix, Linux and Windows systems. The UTM cluster files listed above are generated together with the KDCFILE.
(CLUSTER,ROOTSRC)	This parameter value is only supported on Unix, Linux and Windows systems. The UTM cluster files listed above are generated together with the ROOT table source.

(CLUSTER,KDCFILE,ROOTSRC)	This parameter value is only supported on Unix, Linux and Windows systems. The UTM cluster files listed above are generated together with the KDCFILE and the ROOT table source.
NO	The parameters are only checked.
ALL	The KDCFILE and the ROOT table source are generated.

i If a ROOT table source is generated, the ROOT statement must be specified. This is the case with the following specifications:

- ROOTSRC
- (KDCFILE,ROOTSRC)
- ALL
- (CLUSTER,ROOTSRC)
- (CLUSTER,KDCFILE,ROOTSRC)

The following must further be noted for Unix, Linux and Windows systems:

- KDCDEF generates the ROOT table source as a C/C++ program under the name *rootname* `.c` (see the ROOT statement) in the directory *filebase* (see the MAX ...,KDCFILE=*filebase...* statement).
- If a KDCFILE is generated for a UTM cluster application, then the cluster user file must already exist. This file is evaluated and extended if required.

GEN-RSA-KEYS =	Specifies whether RSA keys are to be created.
YES	KDCDEF is to generate RSA keys. RSA keys are required by applications in which objects (TAC, PTERM or TPOOL) are generated with an encryption level. If GEN-RSA-KEYS=YES then KDCDEF always generates RSA keys for password encryption irrespective of the type of objects generated. If GEN-RSA-KEYS=YES is set but the encryption functions are not available then KDCDEF issues the warning message K508. However, the KDCFILE is still generated and the application can be operated (without encryption). Default: YES

NO

KDCDEF is not to generate RSA keys.

GEN-RSA-KEYS=NO should not be used unless

- openUTM is being run without the encryption functionality ,
- or, after the KDCDEF run, the RSA keys are transferred from an old KDCFILE to the new KDCFILE using KDCUPD. For more information on transferring RSA keys with KDCUPD see "The tool KDCUPD - updating the KDCFILE".

If objects with encryption are generated in an application and if no RSA keys are available, the application can run but with certain restrictions, i.e.:

- TACs with encryption level cannot be called,
- no connection can be established to PTERMs or TPOOLs generated with encryption level.

ROOTSRC=

filename

This parameter is only supported on BS2000 systems.

This parameter is significant only when generating the ROOT table source.

filename can be up to 54 characters in length.

A ROOT source with the CSECT name *rootname* is generated and stored in the KDCROOT file *filename*. *rootname* is defined in the ROOT statement.

Default: ROOT.SRC.ASSEMB.*rootname*

6.5.34 OSI-CON - define a logical connection to an OSI TP partner

The OSI-CON control statement allows you to assign a real partner application to an OSI-LPAP partner for communication based on the OSI TP protocol. It is used to define logical connections between the local UTM application and a partner application. For this purpose, you must specify:

- the name of the OSI TP access point in the local application, via which the connection is to be established. This is defined using the ACCESS-POINT statement.
- the address of the OSI TP access point of the partner application. This address consists of P-selector, S-selector, T-selector and N-selector and optionally the port nummer (parameter LISTENER-PORT).

  On Unix, Linux and Windows systems the following operands are used to describe the T-selector:

  - TRANSPORT-SELECTOR (=address of the partner application on the partner computer)
  - T-PROT (the transport protocol used)
  - TSEL-FORMAT (format identifier of the T-selector)
  - LISTENER-PORT (port number for RFC1006)

  See "Providing address information for the CMX transport system (Unix, Linux and Windows systems)" for more information.

The partner application sets up the connection to the local application via an OSI-LPAP partner, which is defined in the OSI-LPAP statement. Here you generate the number of connections, the names of the individual connections, and so on. The communication parameters of the OSI-LPAP partner are assigned to the OSI-CON statement using the operand OSI-LPAP=*osi_lpap_name*. The logical connection is thus generated in a single OSI-CON statement, even if there are several parallel connections to the partner application.

If a partner application can be accessed in various remote systems at different times, you must define several addresses and thus replacement connections for the OSI-LPAP partner assigned to this application. For generation purposes, this involves assigning several OSI-CON statements (OSI-CON statements with the same *osi_lpap_name* and LOCAL-ACCESS-POINT) to a single OSI-LPAP statement (see "OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP"). However, only one OSI-CON statement can be active at any one time. You can switch to a replacement connection by means of administration.

When you use OSI-LPAP bundles, then the following also applies to the OSI-CONs of the slave LPAPs in a LPAP bundle:
All OSI-CONs of all slave LPAPs in a LPAP bundle must be assigned the same access point (see also section "MASTER-OSI-LPAP - Defining the master LPAP of an OSI-LPAP bundle").

```
OSI-CON  connection_name
         [ ,ACTIVE={ YES | NO } ]
           ,LOCAL-ACCESS-POINT=access_point_name
           ,NETWORK-SELECTOR=C'c'
           ,OSI-LPAP=osi_lpap_name
           ,PRESENTATION-SELECTOR={ *NONE |
                                    (C'c' [, STD | EBCDIC | ASCII ]) |
                                    X'xx' }
           ,SESSION-SELECTOR={ *NONE |
                               (C'c' [, STD | EBCDIC | ASCII ]) |
                               X'xx' }
           ,TRANSPORT-SELECTOR=C'c'
         [ ,LISTENER-PORT=number ]

         additional operands on Unix, Linux and Windows systems
         [ ,MAP={ USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4 } ]
         [ ,T-PROT= RFC1006 ]
         [ ,TSEL-FORMAT={ T | E | A } ]
```

connection_name — Name of the logical connection between the local UTM application and the partner application for communication based on the OSI TP protocol.
connection_name identifies the connection in the local application. It can be up to eight characters in length and must be unique in the local application.

ACTIVE= — Status (active or inactive) of the logical connection to the partner application. In the case of replacement connections to the partner application, several OSI-CON control statements are issued with the same *osi_lpap_name* of an OSI-LPAP partner. However, only one OSI-CON statement can be generated with ACTIVE=YES. All others must be defined with ACTIVE=NO. You can then switch to the replacement connections by means of administration.

YES — The connection defined in this OSI-CON statement is active.

Default: YES

NO — The connection defined in this OSI-CON statement is inactive.

LISTENER-PORT= number

Port number of the partner application.
All port numbers between 1 and 65535 are allowed.

Default: 0 (no port number)

On BS2000 systems, the transport system uses the standard port 102 in this case.
On Unix/Linux and Windows systems, a port number must be specified.

LOCAL-ACCESS-POINT=access_point_name

Name of the local OSI TP access point used for communication with the partner application. This is defined using the ACCESS-POINT control statement.

If replacement connections (several OSI-CON statements with the same *osi_lpap_name*) have been defined for the OSI-LPAP partner to which the partner application is assigned, the same local access point must be specified for all replacement connections.

If the application context of the OSI-LPAP partner uses the CCR syntax, the following address components must also be defined for the local access point:

- APPLICATION-ENTITY-QUALIFIER (see the description of the ACCESS-POINT statement in section "ACCESS-POINT - create an OSI TP access point")
- APPLICATION-PROCESS-TITLE (see the description of the UTMD statement in section "UTMD - application parameters for distributed processing")

MAP= This operand is only valid on Unix, Linux and Windows systems.

Controls the code conversion (EBCDIC <-> ASCII) for user messages exchanged between partner applications (OSI-LPAP) in the abstract syntax UDT.

User messages are passed in the message area on the KDCS interface in the message handling calls (MPUT/FPUT/DPUT).

For user messages with a different abstract syntax than UDT - i.e. if KCMF does not contain any blanks - UTM does not perform a conversion regardless of the value generated here.

USER openUTM does not convert user messages, i.e. the data in the KDCS message area is transferred between the partner applications unchanged.

Default: USER

SYSTEM | SYS1 | SYS2 | SYS3 | SYS4

UTM converts the user messages based on the conversion tables provided for the code conversion (see section “Code conversion”), i.e.:

- Prior to sending, the code is converted from ASCII to EBCDIC.
- After receipt, the code is converted from EBCDIC to ASCII.

The specifications SYSTEM and SYS1 are synonymous.

The prerequisite is that the message has been created using the abstract syntax of UDT (KCMF = blanks).

UTM assumes that the messages contain only printable characters.

NETWORK-**SEL**ECTOR=C’c’

Name of the partner computer.

The complete name (FQDN) by which the computer is known in the DNS must be specified.

The name can be up to 64 characters long.

N-SEL is a mandatory operand.

No distinction is made between uppercase and lowercase notation; KDCDEF always converts the name of the partner computer into uppercase.

> Please note that the name pair (TRANSPORT-SELECTOR, NETWORK-SELECTOR) specified here must not be identical to the name pair (*remote_appliname*, PRONAM) defined in a CON statement ("CON - define a connection for distributed processing based on LU6.1") , or to the name pair (*ptermname*, PRONAM) defined in a PTERM statement ("PTERM - define the properties of a client/printer and assign an LTERM partner").

OSI-LPAP= osi-lpap_name

Name of the OSI-LPAP partner defined as the logical access point for the partner application in the local application.
osi_lpap_name must be defined in a OSI-LPAP statement.
osi_lpap_name can be up to eight characters in length.

PRESENTATION-**SEL**ECTOR=

Presentation selector of the partner application. This is the address component of the OSI TP access point in the remote partner's system. The specified value must match the presentation selector defined for this access point in the partner application.

*NONE A symbolic presentation selector is not defined.

C'c' The presentation selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The presentation selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted:

STD The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix, Linux and Windows systems = ASCII).

Default: STD

EBCDIC The characters are interpreted as EBCDIC code.

ASCII The characters are interpreted as ASCII code.

X'x' The presentation selector is entered in the form of a hexadecimal number (x). The value specified for *x* can be up to 32 hexadecimal digits (corresponds to16 bytes) in length. You must enter an even number of hexadecimal digits.

SESSION-**SEL**ECTOR=

Session selector of the partner application. This is the address component of the OSI TP access point in the remote partner's system. The specified value must match the session selector defined for this access point in the partner application.

*NONE — A session selector is not defined.

C'c' — The session selector is entered in the form of a character string (c). The value specified for *c* can be up to 16 characters in length. The session selector is case-sensitive.

In the case of a character string, you can chose the code in which the characters are interpreted:

STD — The characters are interpreted as a machine-specific code (BS2000 = EBCDIC; Unix, Linux and Windows systems = ASCII).

Default: STD

EBCDIC — The characters are interpreted as EBCDIC code.

ASCII — The characters are interpreted as ASCII code.

X'x' — The session selector is entered in the form of a hexadecimal number (x). The value specified for *x* can be up to 32 hexadecimal digits (corresponds to 16 bytes) in length. You must enter an even number of hexadecimal digits.

TRANSPORT-**SEL**ECTOR=C'c'

You can enter up to eight printable characters. Permitted characters include uppercase letters, numbers, and the special characters $, # and @. Hyphens are not permitted. The first character must be an uppercase letter.

T-SEL= is a mandatory operand.

In T-SEL= you must specify the following:

- *BS2000 systems:* The BCAM application name of the remote partner.
- *Unix, Linux and Windows systems:* T-selector of the partner application.

  You must specify the T-selector in T-SEL that is assigned to the partner application in the remote system for CHECK-RFC1006=YES.

> **i** Please note that the name pair (TRANSPORT-SELECTOR, NETWORK-SELECTOR) specified here must not be identical to the name pair (*remote_appliname*, PRONAM) defined in a CON statement (in section "CON - define a connection for distributed processing based on LU6.1"), or to the name pair (*ptermname*, PRONAM) defined in a PTERM statement (in section "PTERM - define the properties of a client/printer and assign an LTERM partner").

T-PROT= The address format with which the OSI TP partner signs on to the transport system.

Information on the following address formats can be found in section "PCMX documentation" (openUTM documentation).

RFC1006 Address format RFC1006
ISO transport protocol based on TCP/IP and RFC1006 convergence protocol.

Default: RFC1006

TSEL-FORMAT= The format identifier of the T-selector
The format identifier specifies the coding of the T-selector in the transport protocol. You will find more information in section "PCMX documentation" (openUTM documentation).

T TRANSDATA format

E EBCDIC format

A ASCII format

Default:

- T If the character set of the value of T-SEL corresponds to the TRANSDATA format
- E Otherwise

It is recommended to explicitly specify a value for TSEL-FORMAT operation via TCP/IP with RFC1006.

6.5.35 OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP

The OSI-LPAP control statement allows you to define a logical access point in the local application for a partner application with which you wish to communicate on the basis of the OSI TP protocol. This logical access point is known as an OSI-LPAP partner. For each OSI-LPAP partner, you must define a logical partner name and the following logical connection properties:

- The application entity title (AET) of the partner application. This must be defined if you are working with transaction management (commit functional unit) or if a heterogeneous partner requires an AET in order to establish a connection. The AET consists of the following components, which must be specified for the partner application:
  - the application entity qualifier (AEQ) of the remote access point (see the description of the ACCESS-POINT statement on "ACCESS-POINT - create an OSI TP access point")
  - The application process title (APT) of the partner application (see the description of the UTMD statement on "UTMD - application parameters for distributed processing")
- The application context used for communication with the partner application based on the OSI TP protocol. If you are not using a standard application context, you define your application context using the APPLICATION-CONTEXT statement (see "APPLICATION-CONTEXT - define the application context"). If the application context of the OSI-LPAP partner contains the CCR syntax, an AEQ and an APT must be specified for the partner application.
- The number and properties of connections to the partner application
- Access rights of the partner application in the local applicationThe operands KSET and ASS-KSET are provided to define the access rights. In KSET you specify the highest level of access rights of the OSI TP partner that the OSI TP partner will have when it signs on to the local application with a user ID. You can restrict these access rights with the ASS-KSET operand. The restricted access rights take effect when the OSI TP partner does not pass a user ID when signing on, i.e. the "association user" is active.
- Administration authorization of the partner application in the local application
- Maximum values for the message queue of the OSI-LPAP partner.

If a communication partner can be accessed in various remote systems at different times, you can assign several addresses to this partner. This involves assigning several OSI-CON statements (with the same *osi_lpap_name*, see "OSI-CON - define a logical connection to an OSI TP partner") to a single OSI-LPAP statement. However, only one OSI-CON statement can be active at any one time. You can switch to a replacement connection by means of administration. All OSI-CON connections belonging to an OSI-LPAP partner must have the same local access point.

OSI-LPAP	osi_lpap_name
	  ,APPLICATION-CONTEXT=application_context_name
	[,**A**PPLICATION-**E**NTITY-**Q**UALIFIER=application_entity_qualifier
	  ,**A**PPLICATION-**P**ROCESS-**T**ITLE=object_identifier]
	[,ASS-KSET=keysetname2]
	  ,**ASS**OCIATION**-NAMES**=association_name
	[,ASSOCIATIONS=number]
	[,BUNDLE=master-lpap-name]
	[,CONNECT=number]
	  ,CONTWIN=number
	[, DEAD-LETTER-Q={ NO | YES }]
	[,IDLETIME=time]
	[,KSET=keysetname1]
	[,PERMIT={ ADMIN | SATADM[1] | (ADMIN,SATADM)[1] }]
	[,QLEV=number]
	[,STATUS={ ON | OFF }]
	[,TERMN=termn_id]

[1]only permitted on BS2000 systems

osi_lpap_name

Name of the OSI-LPAP partner of the partner application, which is used by the program units of the local UTM application to address the partner application. *osi_lpap_name* can be up to eight characters in length.

osi_lpap_name must be unique and must not be assigned to any other object in name class 1. See also section "Uniqueness of names and addresses".

APPLICATION-CONTEXT=application_context_name

Name of the application context to be used by the partner application.

This is a mandatory operand.

By default, openUTM supports the following application contexts:

- UDTAC
- UDTDISAC
- XATMIAC
- UDTCCR
- UDTSEC
- XATMICCR

Further information can be found in the description of the APPLICATION-CONTEXT statement on "APPLICATION-CONTEXT - define the application context". If the generated application context does not match that used by the partner application, openUTM rejects the connection request with the following UTM messages:

```
P001 APPLICATION CONTEXT NOT SUPPORTED        or
P011 Abstract syntax not permitted
```

APPLICATION-**E**NTITY-**Q**UALIFIER=application_entitiy_qualifier

Application entity qualifier of the partner application. This is combined with the application process title to address a partner application when using a heterogeneous link or working with transaction management (commit functional unit). *application_entitiy_qualifier* is a positive integer used to call the partner application in the remote system.

If the application context contains the CCR syntax, this is a mandatory operand. The name pair *application_entity_qualifier* and *object_identifier* must be unique within the UTM application.

The *application_entity_qualifier* specified here must be assigned to access point in the partner application.

Minimum value: 1
Maximum value: 67108 863 (2^{26}-1)

APPLICATION-**P**ROCESS-**T**ITLE=object_identifier

The application process title of the partner application is to be specified as the *object_identifier*. The application process title is combined with the application entity qualifier to address a partner application when using a heterogeneous link or working with transaction management (commit functional unit).

If the partner application is a UTM cluster application then the application process title (APT) of a node application must be specified here. Here it should be noted that when a node application is started, openUTM may extend the APT generated for this application by the node index of the application. See also the description of the APPLICATION-PROCESS-TITLE operand in the UTMD statement in section "UTMD - application parameters for distributed processing".

If the application context contains the CCR syntax, this is a mandatory operand. The name pair *application_entity_qualifier* (of the OSI-LPAP statement) and *object_identifier* must be unique within the UTM application.

For information on defining the APT, see the UTMD statement in section "UTMD - application parameters for distributed processing".

ASS-KSET= ksetname2

ASS-KSET is only allowed if the local application is generated with user IDs. You may only set ASS-KSET in conjunction with KSET.

You must specify the name of the key set in *ksetname2*. The key set must be defined with a KSET statement.

You specify the minimum access rights that the partner application can have in the local application with ASS-KSET= .

The key set specified in *ksetname2* takes effect when the partner application does not pass a user ID to openUTM when establishing the association. The access rights result from the set of key codes contained in the key set generated with KSET= and with ASS-KSET= (intersection of the sets). For this reason, all key codes contained in ASS-KSET=*ksetname2* should also be contained in KSET=*ksetname1*.

Default: No key set
The access rights specified in KSET are always valid.

ASSOCIATION**-NAMES**=association_name

Name defined in the local application for logical connections to the partner application.

Connection names consists of the value of *association_name* as a prefix, followed by a serial number between 1 and the value of the ASSOCIATIONS operand, i.e. the number of parallel connections. The entire name can be up to eight characters in length. The maximum length of *association_name* depends on the value specified for ASSOCIATIONS. The following applies for the number of connections:

Number of decimal places in the value specified for ASSOCIATIONS +number of characters in *association_name* <= 8

Example

If ASSOCIATIONS=10 and
ASSOCIATION-NAMES=ASSOC,

The connection names are ASSOC01, ASSOC02,...,ASSOC10.

These are used as association names in the local UTM application. The same name must not be defined for a user ID (USER) or a session name for distributed processing based on LU6.1 (LSES).

ASSOCIATIONS= number

Maximum number of parallel connections to the partner application. This depends on layers 1 - 6 of the OSI reference model defined by ISO (in particular on layer 4, the transport layer).

The number of parallel connections must be coordinated with the generation of the partner application.

Default: 1
Minimum value: 1
Maximum value: 21000
The maximum number of associations of all OSI-LPAP statements in a UTM application is restricted by the size of the name space of the UTM application (see "Number of names").

BUNDLE= master-lpap-name

Name of a master LPAP. By specifying *master-lpap-name*, this OSI-LPAP partner becomes a slave LPAP of the corresponding master LPAP.

The master LPAP specified here must be generated with a MASTER-OSI-LPAP statement.

CONNECT= number

Number of connections to be established automatically with the partner application when the local application is started. Automatic connection setup can be requested in either the local application or the partner application. The connection is established as soon as both partners are available.

The following applies on BS2000 systems: If the partner application runs on Unix, Linux or Windows systems, a value greater than 0 should be generated only if the BS2000 system is configured in such a manner that connections to this partner can be actively established. See also section "Coordinating the UTM and BCAM generations (BS2000systems)".

Default: 0 Maximum value:
Number of parallel connections specified in the ASSOCIATIONS operand.

CONTWIN= number

Number of connections for which the local application is to act as the contention winner. The local application is the contention loser for all other connections (ASSOCIATIONS= entry minus CONTWIN= entry).

The contention winner of a connection is responsible for managing that connection. However, jobs can be started both by the contention winner and by the contention loser. If both communication partners attempt to initiate a job simultaneously, the connection is reserved by the contention winner job.

This is a mandatory operand.

The number of contention winners and contention losers must be coordinated with the generation of the partner application.
The contention winner should be the communication partner that initiates jobs most frequently. If this application cannot set up any connections to the partner application, e.g. because the BCMAP entries are missing, this application can nevertheless be the contention winner provided the partner application sets up all the connections automatically.

Minimum value: 0
Maximum value: Number of parallel connections specified for the ASSOCIATIONS operand.

DEAD-LETTER-Q= specifies whether asynchronous messages to this LPAP partner that could not be sent due to a permanent error are to be saved in the dead letter queue.

Monitoring of the number of messages in the dead letter queue is enabled and disabled with the MAX ...,DEAD-LETTER-Q-ALARM statement.

YES Asynchronous messages to this LPAP partner that could not be sent due to a permanent error are saved in the dead letter queue provided (in the case of message complexes) no negative confirmation job has been defined.

NO No asynchronous messages to this LPAP partner are saved in the dead letter queue.

Default: NO

> **i** Main jobs for message complexes (MCOM) with negative confirmation jobs are never saved in the dead letter queue as the negative confirmation jobs are activated in case of errors.
>
> If the number of messages in the dead letter queue is limited with QLEV, messages may be lost in the event of errors. If the number is not limited, the openUTM page pool generated must be sufficiently large. If there is a threat of a page pool bottleneck, the dead letter queue can be locked during application run with STATUS=OFF.

IDLETIME= time

Number of seconds for which the idle state of a connection is monitored. If the connection is not reserved by a job within the period specified in *time*, openUTM shuts down the connection.

IDLETIME=0 means that the idle state of the connection is not monitored.

Default: 0
Minimum value: 60
Maximum value: 32767

If you specify a value that is greater than zero and smaller than the minimum value, KDCDEF replaces the value with the minimum value.

KSET= keysetname1

Specifies the maximum access rights of the partner application in the local application. The name of a key set is to be specified in *keysetname1*. The key set must be defined with a KSET statement.

If the OSI TP partner does not pass a user ID to the local application for an OSI TP dialog, then its access rights for this OSI TP dialog result from the set of key codes that are in the key set generated with KSET= as well as with ASS-KSET= (intersection).

The key set *keysetname1* should therefore also contain all key codes that are in the key set generated with ASS-KSET= .

If the OSI TP partner does pass a user ID, then its access rights for this OSI TP dialog result from the set of key codes that are contained in the key set of the user ID as well as in the key set of the OSI-LPAP generated with KSET.

Default: No key set,
i.e. only those services can be started or remote services (LTAC) generated in the local application can be addressed that are not secured with a lock code.

PERMIT= Authorization level of the partner application

ADMIN The partner application can execute administration functions in the local application.

SATADM The partner application can execute SAT preselection functions in the local application, i.e. it can activate and deactivate the SAT logging of certain events (UTM SAT administration authorization).

(ADMIN,SATADM)

The partner application can execute administration and SAT preselection functions in the local application.

Default:
If the operand is not specified, the partner application cannot execute administration and SAT preselection functions in the local application.

QLEV= queue_level_number

Maximum number of asynchronous messages that can be accommodated in the message queue of the OSI-LPAP partner. If this threshold value is exceeded, further APRO-AM calls to this LPAP partner are rejected with UTM message 40Z.

Default: 32767
Minimum value: 0
Maximum value: 32767 (i.e no restriction of the queue length)

STATUS= Specifies whether the OSI-LPAP partner is locked. This status can be modified by the administrator using the KDCLPAP administration command.

ON The OSI-LPAP partner is unlocked. Connections between the partner application and the local application can be established or may be already in place.

Default: ON

OFF The OSI-LPAP partner is locked. Connections cannot be established between the partner application and the local application.

TERMN= termn_id

Identifier up to two characters in length, which indicates the type of communication partner. *termn_id* is not queried by openUTM, but is used by the user when querying or grouping terminal types, for example. *termn_id* is entered in the KB header for services, i.e. for services started by the partner application in the local application.

Default: A6

6.5.36 PROGRAM - define a program unit

The PROGRAM control statement allows you to define the name and properties of a program unit.
If a ROOT table source is to be generated in the KDCDEF run (OPTION statement with GEN=ROOTSRC or GEN=ALL), then you must issue at least one PROGRAM statement.

Generating UTM program units on BS2000 systems

```
PROGRAM  objectname
         , COMP={ ASSEMB    |
                  C         |
                  COB1      |
                  FOR1      |
                  PASCAL-XT |
                  PLI1      |
                  SPL4      |
                  ILCS }
         [ ,LOAD-MODULE=lmodname ]
```

objectname

Access point for a program unit (CSECT or ENTRY name). *objectname* may be up to 32 characters in length.

For details on the characters allowed refer to the section "Format of names".

COMP=

Designates the runtime system that the program unit will be used for.

This is a mandatory operand.

You must specify COMP=ILCS for all program units that support ILCS (Inter Language Communication Services), e.g. program units under COBOL85, FORTRAN90, C, etc. Whether or not ILCS is supported depends on the compiler version used and on the runtime system version under which the program unit runs.

The value that you must specify for COMP can be found in the appendix of the openUTM manual “Using UTM Applications on BS2000 Systems”.
Please consider these notes especially if the programs were compiled with an older compiler version.

> COMP=C is a synonym for COMP=ILCS.
> The KDCADM administration program must be generated with COMP=ILCS and the KDCSHUT transaction code must be assigned at least with a TAC statement.

LOAD-MODULE= lmodname

LOAD-MODULE identifies the name of the load module in which the program unit was linked. This load module must be defined using the LOAD-MODULE statement. *lmodname* can be up to 32 characters in length.This name is subject to the same rules as the element names of a program library (see also section "Format of names").

Please note the following when using the LOAD-MODULE operand:

- The KDCADM administration program must not be assigned to a load module generated with LOAD-MODE=ONCALL in the LOAD-MODULE statement.

Generating UTM program units on Unix, Linux or Windows systems

PROGRAM	objectname ,COMP={ C | COB2 | CPP | MFCOBOL | NETCOBOL } [,SHARED-OBJECT=shared_object_name]

objectname — Name of the access point of the program unit. The name must be alphanumeric and may be up to 32 characters in length. For details on the characters allowed refer to the section "Format of names".

COMP= — Designates the compiler used to compile the program unit.

C — C compiler

Default: C

CPP — C++ compiler

COB2 — COBOL compiler (Server Express / NetExpress / Visual COBOL)
This generates a COBOL program that was compiled using a COBOL complier from Micro Focus.
Only numbers and uppercase letters can be used for the PROGRAM-ID and the access points. This not only complies with IBM conventions, but also guarantees the portability of the programs.

MFCOBOL — COBOL compiler (Server Express / NetExpress / Visual COBOL), has the same effect as COB2. I.e. a COBOL program that was compiled using a Micro Focus Cobol compiler is generated.

NETCOBOL — NetCOBOL compiler from Fujitsu.

This parameter value is supported only on Unix and Linux systems.

A COBOL program that was compiled using the Fujitsu NetCOBOL compiler is generated.

> **CAUTION!**
> In a UTM application, programs must not be simultaneously generated with MFCOBOL/COB2 and NETCOBOL!

SHARED-OBJECT= shared_object_name

(Program exchange using the dynamic linker)
This operand need only be specified if the program unit is to be loaded dynamically. *shared_object_name* is the name of the shared object (Unix and Linux system) or DLL (Windows system) into which the program unit was incorporated. This shared object/DLL must be defined using the SHARED-OBJECT statement.

6.5.37 PTERM - define the properties of a client/printer and assign an LTERM partner

The PTERM control statement allows you to define the properties of a physical client or printer of the UTM application.

Clients are terminals, UPIC clients and transport system applications. Transport system applications (for short TS application) are understood to be all applications that are generated as PTYPE=APPLI or PTYPE=SOCKET. See also the PTYPE operand in the table in section "PTERM - define the properties of a client/printer and assign an LTERM partner" (BS2000 systems) or "PTERM - define the properties of a client/printer and assign an LTERM partner" (Unix, Linux and Windows systems).

You must always issue a PTERM statement for clients when connections to the client or printer are to be established from the local UTM application.

The PTERM statement allows you to assign an LTERM partner defined using the LTERM statement to the client /printer. A separate LTERM statement must be written for each client or printer (see also "LTERM - define an LTERM partner for a client or printer" for more information on this subject).

If desired, you can first define the client/printer in a PTERM statement and then assign it to an LTERM partner later on during operation by means of dynamic administration. Exceptions are UPIC clients and TS applications. You need to assign an LTERM partner to these immediately.

If LTERM pools have not been generated (TPOOL statement, see "TPOOL - define an LTERM pool"), you must assign a client in the LTERM= operand of at least one PTERM statement. Only then can a connection be established in order to access the application.

> Printers are not supported by openUTM on Windows systems.

Address of the client or printer

For the application to be able to establish connections to the partner application, you must specify the partner address. The following operands are used to do this:

- *ptermname* (name/T-selector of the communication partner)
- PRONAM (name of the host partner)
  On Unix, Linux and Windows systems, the name of the partner processor may only be specified if the partner is a remote UPIC client (UPIC-R) or a TS application (PTYPE= APPLI or SOCKET).
- LISTENER-PORT (TCP/IP port number).
  On BS2000 systems, LISTENER-PORT may only be specified with PTYPE=APPLI and PTYPE=SOCKET.

The following operands are used for further definition of the partner address on Unix, Linux and Windows systems:

- T-PROT (address format for the transport protocol used)
- TSEL-FORMAT (format identifier of the T-selector)

See section "Providing address information for the CMX transport system (Unix, Linux and Windows systems)" for more information or section "Providing address information for the SOCKET transport system (Unix, Linux and Windows systems)".

If the connection to a TS application is to be established via the socket interface with native TCP/IP as the transport protocol, then you must specify the computer on which the TS application will run in PRONAM and the port number on the host partner on which the TS application waits for requests to establish a connection from the network in LISTENER-PORT. You must specify an application name that was generated for T-PROT=SOCKET in BCAMAPPL (see also "BCAMAPPL - define additional application names").
You can specify whether or not openUTM is to handle code conversion in the MAP operand.

Uniqueness of names

When generating the CON, PTERM and MUX statements, please note that the name triplet (*appliname* or *ptermname*, *processorname*, *local_appliname*) must be unique within the generation run.

> In order to make the generation of your UTM application more independent of the terminal type, it is possible to incorporate terminals in the configuration without explicitly specifying their type. For this purpose, set the PTYPE operand to *ANY. During connection setup, openUTM then takes the partner type (PTYPE) from the user services protocol (connection letter) and checks whether or not this type is supported. If not, openUTM rejects the connection request.

PTERM	ptermname [,BCAMAPPL=local_appliname] [,CONNECT={ YES | NO }] [,ENCRYPTION-LEVEL={ NONE | 3 | 4 | 5 [2] | TRUSTED }] [,IDLETIME=time] [,LISTENER-PORT=number] [,LTERM=ltermname] [,MAP={ USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4 }] ,PRONAM={ processorname | C'processorname'| *RSO [1] } *only mandatory on BS2000 systems* [,STATUS={ ON | OFF }] [,TERMN=termn_id] [,USP-HDR={ NO | MSG | ALL }] ***BS2000, Unix and Linux system specific operand*** [,CID=printer_id] ***BS2000 specific operands*** [,PROTOCOL={ N | STATION }] ,PTYPE={ partnertyp | *ANY | *RSO } [,USAGE={ D | 0 }] ***Unix, Linux and Windows system specific operands*** [,PTYPE={ partnertyp | PRINTER|(PRINTER ,printertype [,class])}] [,T-PROT=RFC1006 | SOCKET] [,TSEL-FORMAT={ T | E | A }]

[1]This operand value is only permitted on BS2000 systems

[2] These operand values are only permitted on Unix, Linux, and Windows systems

ptermname

Name of the client or printer up to 8 characters in length

The specified name must be unique and must not be assigned to any other object in name class 3. See also section "Uniqueness of names and addresses"

The following cases can arise:

Socket applications (PTYPE=SOCKET)

- If the connection is to be established from the local application to the client, then any *ptermname* can be selected. It is only relevant internally in UTM then, e.g. for administration.
- If the connection is to be established externally (initiated by the client), then *ptermname* must contain the port number via which the client addresses the UTM application. You must then specify the prefix “PRT“ followed by 5 digits (with leading zeros, if necessary) that designate the port number as the *ptermname*. For example, you must specify *ptermname*=PRT08050 if the client is to address the UTM application via the port 8050.

  Establishing the connection externally to a specific PTERM is only possible by partners that set their port numbers themselves when establishing a connection. openUTM does not do this, i.e. you cannot issue any PTERM statements for a remote UTM application that is to establish SOCKET connections to the local application. In this case, you need to connect via an LTERM pool.

BS2000 systems:
When defining an RSO printer (PTYPE=*RSO), you must specify the name of the printer as defined for RSO.

Unix, Linux or Windows systems:
For UPIC clients and TS applications with PTYPE=APPLI you must specify the T-selector that is assigned to the client in the remote system for ptermname if OPTION CHECK-RFC1006=YES.

In the case of printers, ptermname is the name of the spool queue or printer group as defined during generation of the Unix or Linux system. To output the data, the printer process (utmprint) calls the utmlp script (see PTYPE=PRINTER on "PTERM - define the properties of a client/printer and assign an LTERM partner").

In the case of local terminals and pseudo terminals, the result of the command `basename `tty`` must be specified for *ptermname* in each PTERM statement so that the UTM generation matches the terminal generation under the Unix or Linux system.

On Unix and Linux systems, the default *ptermname* assigned by openUTM may not be unique. Depending on the type of network to which the system is connected, it is possible to have two or more pseudo terminals for which the last term of the tty (after the last slash) is identical. Only one terminal can use this *ptermname* to establish a connection with the application. The connection request from the second terminal will be rejected by openUTM.

Example The system contains the ttys `/dev/pts/12` and `/dev/inet/12`. If terminal `/dev/pts/12` requests a connection to the application with the *ptermname* 12 and terminal `/dev/inet/12` is already linked to the application, the connection request issued by `/dev/pts/12` is rejected. You must use the last two parts of the output of the *tty* command as the *ptermname*; e.g. instead of PTERM 12, enter the statements PTERM `pts/12` and PTERM `inet/12`.

You can also generate an LTERM pool with PTYPE=TTY instead.

Windows systems:
For terminals any name can be specified for ptermname.

BCAMAPPL= local_appliname

Name of the local UTM application as defined in MAX ...,APPLINAME= or the BCAMAPPL statement (see also "BCAMAPPL - define additional application names"). When establishing a connection between the client/printer and the UTM application, *local_appliname* must be specified as the partner name. In the case of terminals and printers, the name defined in MAX ...,APPLINAME= must be used here. For PTERMs with PTYPE=SOCKET you must specify a name in *local_appliname* that is generated using BCAMAPPL ... T-PROT=SOCKET.

The BCAMAPPL name specified in the CLUSTER statement is not permitted here.

Default value:
Value in MAX APPLINAME= (primary name of the UTM application). This default value applies if BCAMAPPL= is not specified or contains blanks only.

CID= printer_id

This operand is only supported on BS2000, Unix and Linux systems.

This applies only for printers assigned to a printer control LTERM. *printer_id* is used to identify the printers at the printer control LTERM, and can be up to eight characters in length.

The printer control LTERM is used to manage printers, printer queues and print jobs.

If the printer is assigned an LTERM partner for which a printer control LTERM has been defined using LTERM ...,CTERM=*ltermname2*, the printer itself is also assigned to this printer control LTERM. The combination of *ltermname2* of the printer control LTERM and CID must be unique.

Default: A CID is not assigned to the printer

CONNECT= Specifies whether or not openUTM establishes a connection to the client or printer when starting the application.

- On BS2000 systems CONNECT= is only relevant for TS applications, terminals and printers.
- On Unix and Linux systems CONNECT= is only relevant for TS applications and printers.
- On Windows systems CONNECT= is only relevant for TS applications.

YES

When starting the application, openUTM automatically attempts to establish a connection.

In the case of printers generated with LTERM ...,PLEV > 0, openUTM does not attempt to establish the logical connection until the PLEV value is exceeded.

If a connection cannot be established to a printer or TS application, openUTM makes repeated attempts to establish the connection at intervals defined in MAX ...,CONRTIME.

BS2000 systems:
If a logical connection to a terminal cannot be set up, this can be performed explicitly by the user at a later point in time.

Unix and Linux systems:
When starting the application, openUTM automatically creates a printer process for executing print jobs, which is assigned to *ptermname*.

NO

When starting the application, openUTM does not attempt to establish a connection.

CONNECT=NO must be specified for clients with PTYPE=UPIC-R and UPIC-L.

Default: NO

> *Unix and Linux systems:*
> A printer process is not created for *ptermname*. This can only be achieved by issuing the administration command KDCPTERM ACT=C (or KDCLTERM for the assigned LTERM partner). The printer process can then accept print jobs from work processes, which are directed to the appropriate spool queue.

ENCRYPTION-LEVEL=

Only relevant for UPIC clients that support encryption and on BS2000 systems for some terminal emulations that support encryption also.

In ENCRYPTION-LEVEL you set the minimum encryption level for the communication with the client.
You specify whether or not the UTM application should request encryption of the message on the connection to the client. You can also define the client as a "trusted" client. See also section "Message encryption connections to clients " for more information on encryption.

The client must be a openUTM-Client with the UPIC carrier system and with encryption functions to be able to encrypt data on the connection to the client.

Default values:

TRUSTED is the default value for:

- HTTP clients und USP-Socket applications, which connect via a transport system access point (BCAMAPPL), that is configured with T-PROT=(..., SECURE).
- Local UPIC clients (PTYPE=UPIC-L) on Unix, Linux and Windows systems

Other values for these partners are changed to TRUSTED by KDCDEF without issuing a message.

NONE is the default value for

- all other types of communication partners.

For partners with PTYPE different from UPIC-R, and on BS2000 different from T9763, the values 3, 4, 5 are changed to NONE by KDCDEF without issuing a message.

BS2000 systems:
A prerequisite for the use of encryption on connections between openUTM and terminal emulations is VTSU-B >= V12.0C.

You can specify the following:

NONE

Encryption of the messages exchanged between the client and the UTM application is **not** requested by openUTM by default.
Passwords are transmitted in encrypted form, if both partners support encryption.
Services for which encryption was generated for their service TACs (see ENCRYPTION-LEVEL in the TAC statement starting on "TAC - define the properties of transaction codes and TAC queues") can only be started by this client if the client negotiates encryption when establishing the connection or establishing the conversation.

3 | 4 | 5

Encryption of the messages exchanged between the client and the UTM application is requested by openUTM by default. The value specifies the encryption level. The client cannot connect unless it supports at least this encryption level. Otherwise openUTM rejects connection setup.

The values have the following meaning:

3 Passwords and input/output messages are encrypted using the AES-CBC algorithm. An RSA key with a key length of 1024 bits is used to exchange the AES key.

4 Passwords and input/output messages are encrypted using the AES-CBC algorithm. An RSA key with a key length of 2048 bits is used to exchange the AES key.

5 Input/output messages are encrypted using the AES-GCM algorithm. The AES key is agreed using the Ephemeral Elliptic Curve Diffie-Hellman method (ECDHE). An RSA key with a key length of 2048 bits is used to sign the public server key.

Level 5 is currently only supported by openUTM for LUW platforms.

BS2000 systems:
VTSU encryption is used for VTSU partners. (see manual "Security Handbook for Systems Support")

> **i** If the application is generated with OPTION GEN-RSA-KEYS=NO, no RSA keys are created in the KDCDEF run. In order to use the encryption functions, you must create the required keys using administration facilities (KC_ENCRYPT or WinAdmin or WebAdmin) or transfer them from an old KDCFILE using KDCUPD.

TRUSTED

The client is a "trusted" client.
Messages between the client and the application are not encrypted. A "trusted" client can also start services whose service TACs request encryption (generated with TAC ENCRYPTION-LEVEL= 2 | 5).

TRUSTED should only be selected if the client communication is conducted through a secure connection.

IDLETIME=

time

May only be specified for dialog partners.
In *time* you enter the maximum time in seconds that openUTM may wait for input from the client outside of a transaction, i.e. after the end of a transaction or after signing on. If this time is exceeded, then openUTM clears down the connection to the client. If the client is a terminal, then message K021 is output before the connection is cleared.

This function serves to improve data security: If a user forgets to sign off from the terminal when taking a break or when finishing his or her work on the terminal, then the connection to the terminal or client is automatically cleared down after the wait time has run out. This reduces the chance of someone gaining unauthorized access to the system.

Default: 0 (= no wait time limit)
MAX TERMWAIT=(...,*time2*) is used for terminals (when it is set).
Maximum value: 32767
Minimum value: 60

If you specify a value that is greater than zero and smaller than the minimum value, KDCDEF replaces the value with the minimum value.

LISTENER-PORT= number

Port number for establishing TCP/IP connections

With socket applications (T-PROT=SOCKET) the LISTENER-PORT= is a mandatory operand.

All port numbers between 1 and 65535 are allowed.

BS2000 systems:
On BS2000 systems LISTENER-PORT may only be specified partners with PTYPE=APPLI or PTYPE=SOCKET. In the case of PTYPE=SOCKET, LISTENER-PORT is a mandatory operand.

For *number* you must specify the port number on which the partner application waits for requests to establish a connection, i.e. the port number on the host partner through which the partner application is addressed.

A port number that is not equal to 0 may only be specified if the local application specified in the parameter BCAMAPPL is not generated with T-PROT=NEA.

Default: 0 (i.e. no port number)
In this case, the transport system uses the default port 102.

Unix, Linux and Windows systems:
LISTENER-PORT is only relevant for TS applications (PTYPE=SOCKET or APPLI).

The LISTENER-PORT is used with T-PROT=SOCKET to specify the port number used to address the partner. No other addressing information is necessary.

Default: 0 (no port numbers)
When OPTION CHECK-RFC1006=YES, a port number must be specified in LISTENER-PORT for PTERMs with PTYPE=APPLI.

LTERM= ltermname

Name of the LTERM partner assigned to the client/printer *ptermname*. This name is used by the client/printer to sign on to the UTM application, and can be up to eight characters in length.

The LTERM operand is mandatory for clients with PTYPE=SOCKET, APPLI and UPIC-R.

The LTERM partner assigned to a client/printer can be changed during runtime using the KDCSWTCH administration command, e.g. if the printer fails. However, it is not possible to assign a dialog LTERM partner (LTERM USAGE=D) to a printer.

> **i** For the printer pool function, you must issue several PTERM statements with the same *ltermname*. A printer pool consists of numerous printers assigned to a single LTERM partner (see also section "Generating printer pools"). openUTM then distributes print jobs cyclically to the various printers in the pool.

MAP=

Controls the code conversion (EBCDIC <-> ASCII) for user messages exchanged between communication partners.

User messages are passed in the message area on the KDCS interface in the message handling calls (MPUT/MGET/FPUT/DPUT/FGET).

USER

openUTM does not convert the data of the message area, i.e. the messages are transferred between the partner applications unchanged.
Note that the user message contains the transaction code in TS applications (partners with PTYPE=SOCKET or APPLI). It must be encoded in the form that the receiving system expects, i.e. on BS2000 systems in EBCDIC and in ASCII on Unix, Linux and Windows systems.

Default: USER

SYSTEM | SYS1 | SYS2 | SYS3 | SYS4

This parameter is only permitted for the following partners:

BS2000 systems: partners with PTYPE=SOCKET

Unix, Linux and Windows systems: partners with PTYPE=SOCKET or APPLI

UTM converts the user messages based on the conversion tables provided for the code conversion (see section "Code conversion"), i.e.:

Prior to sending, the code is converted from ASCII to EBCDIC on Unix, Linux and Windows systems and from EBCDIC to ASCII on BS2000 systems.

After receipt, the code is converted from EBCDIC to ASCII on Unix, Linux and Windows systems and from ASCII to EBCDIC on BS2000 systems.

The specifications SYSTEM and SYS1 are synonymous.

UTM assumes that the messages contain only printable characters.

PRONAM= { processorname | C’processorname’ }

Name of the partner computer. The complete name (FQDN) by which the computer is known in the DNS must be specified.

The name can be up to 64 characters long.

If *processorname* contains special characters it must be entered as a character string using C’...’.

BS2000 systems:

This name is defined during generation of the network. Please consult your network administrator. The assignment of *ptermname* to *processorname* must be unique.

If a TS application is described with which the UTM application communicates via the socket interface, then you must specify the symbolic address of the host partner for *processorname*. The association of the symbolic address to the real IP address must be entered in the name service of the local system (in the RDF file). You must not specify an alias of the host.

When defining an RSO printer (PTYPE=*RSO), you must specify *RSO here.

In the case of clients connecting directly via OMNIS, i.e without mulitplex connection, you must specify PRONAM=*omnis-host* where *omnis-host* is the host on which OMNIS is loaded.

This is a mandatory operand.

PRONAM need not be specified if a default value for this operand is defined beforehand using the DEFAULT statement.

Unix, Linux and Windows systems:

On Unix, Linux and Windows systems PRONAM= is permitted only for remote UPIC clients (PTYPE=UPIC-R) and TS applications (PTYPE=SOCKET or APPLI).

For *processorname*, you enter the real host name under which the IP address of the partner computer is entered in the name service of the local system (e.g. the hosts file) see "Specifying computer names in KDCDEF generation". You must not specify alias names of the computer.

No distinction is made between uppercase and lowercase notation; KDCDEF always converts the name of the partner computer into uppercase.

processorname is a mandatory operand for PTYPE=APPLI, SOCKET or UPIC-R.

Default for PTYPE=TTY, UPIC-L or PRINTER: blanks

PROTOCOL= User services protocol used on connections between the UTM application and the client /printer

N — A user services protocol is not used between the UTM application and the client/printer.

PROTOCOL=N must be set for UPIC clients (PTYPE=UPIC-R), TS applications (PTYPE=SOCKET or APPLI) that communicate with the UTM application via the socket interface (native TCP/IP) and for printers accessed via RSO (PTYPE=*RSO).

Clients with PROTOCOL=N cannot sign on to the UTM application via a multiplex connection (MUX statement).

PROTOCOL=N must be specified for clients connecting directly via ONMIS (i.e. without multiplex connection).

If you specify PTYPE=*ANY, openUTM ignores the entry PROTOCOL=N and automatically sets PROTOCOL=STATION.

Default with PTYPE=SOCKET, APPLI, UPIC-R or *RSO.

STATION — The user services protocol (NEABT) is used between the UTM application and the client /printer.

PROTOCOL=STATION must be specified for clients generated with PTYPE=*ANY. In this case, openUTM requires the user services protocol (NEABT) to determine the device type of the client or printer.

For UPIC clients (PTYPE=UPIC-R), TS applications (PTYPE=APPLI or SOCKET) or printers that are addressed via RSO (PTYPE=*RSO), you are only permitted to specify PROTOCOL=N. If you specify PROTOCOL=STATION it will be ignored.

Default with PTYPE!=SOCKET , UPIC-R or *RSO.

PTYPE= — Type of communication partner

PTYPE on BS2000 systems:

This is a mandatory operand on BS2000 systems.

For PTYPE you must specify the partner type *partnertyp* of the client or printer, the value *ANY, or the value *RSO for RSO printers.

partnertyp — Type of communication partner, i.e. type of client or printer. The value specified in *partnertyp* must match to the type that has been set in the terminal emulation used, for example. The partner type must be entered either in the PTYPE parameter here, or using a DEFAULT statement. The following partner types are supported:

Partner	PTYPE	TERMN
DSS 9748	T9748 [1)]	FE
DSS 9749	T9749	FE
DSS 9750	T9750[1)]	FE
DSS 9751	T9751	FE
DSS 9752	T9752	FF

DSS 9753	T9753	FE
DSS 9754	T9754	FI
DSS 9755	T9755 [2)]	FG
DSS 9756	T9756[2)]	FG
DSS 9763	T9763	FH
DSS 9770	T9770	FK
DSS 9770R	T9770R	FK
FHS-DOORS Front End	DSS-FE	FH
DSS 3270 (IBM)	T3270	FL
DSS X28 (TELETYPE)	THCTX28	C5
DSS X28 (VIDEO)	TVDTX28	C6
FHS-DOORS Front End	DSS-FE	FH
Data station PT80	TPT80	C4
9001 printer	T9001	C7
9002 printer	T9002	FA
9003 printer	T9003	F9
9004 printer	T9004	FD
9001-3 printer	T9001-3	CA
9001-893 printer	T9001-893	CB
9011-18 printer	T9011-18	CC
9011-19 printer	T9011-19	CD
9012 printer	T9012	CE
9013 printer	T9013	C9
9021 printer	T9021	CH
9022 printer	T9022	CF
3287 printer	T3287	CG

Transport system application that is not a socket application, e.g.: DCAM, CMX or UTM application.	APPLI	A1
Intelligent terminal	THOST	A3
UPIC client	UPIC-R	A5
USP-Socket application	SOCKET	A7
HTTP client	SOCKET	A8
Secure USP-Socket application	SOCKET	A9
HTTPS client	SOCKET	AA

[1)]The PTYPEs T9748 and T9750 refer to the same terminal type.

[2)]The PTYPEs T9755 and T9756 refer to the same terminal type.

The VTSU version in which the individual terminals are supported can be found in the respective DCAM, FHS and TIAM manuals. If a terminal is not supported by VTSU, openUTM rejects connection requests from this terminal and outputs UTM messages K064 and K107.

*ANY

A PTYPE=*ANY entry generates a VTSU client. The client/printer is incorporated in the configuration without precise information on the device type. During connection setup, openUTM takes the device type from the user services protocol. Only then can it be determined whether or not the partner type is supported.

The advantage of PTYPE=*ANY is that it allows you to include clients in the configuration without having to know how their type. The configuration is also easier to maintain, because even if the type is modified in the terminal emulation, for example, this client can still sign on to the application without having to modify the KDCDEF generation.

If terminal mnemonics (TERMN operand) are not explicitly generated for clients defined with PTYPE=*ANY, the default terminal mnemonic of the partner type is used for connection setup.

*RSO

If PTYPE=*RSO, support is provided for printers via RSO. Instead of establishing a transport connection, openUTM reserves the printer in RSO and transfers the message to be printed to RSO.

PTYPE on Unix, Linux and Windows systems:

partnertyp

Type of communication partner, i.e. type of client or printer. For *partnertyp* you can specify the following:

Partner	**PTYPE**	**TERMN**
Terminal	TTY	F1
PRINTER *(only on Unix and Linux systems)*	PRINTER	F2
PRINTER, printertype, class *(only on Unix and Linux systems)*	PRINTER	F3
Transport system application that does not use the socket interface (for example UTM, CMX or DCAM application).	APPLI	A1
local UPIC client	UPIC-L	A2
remote UPIC client	UPIC-R	A5
USP socket application	SOCKET	A7
HTTP client	SOCKET	A8
secure USP socket application	SOCKET	A9
secure HTTP client	SOCKET	AA

Default: TTY

PRINTER

Printer without additional parameters.
To output the data, the printer process (*utmprint*) calls the *utmlp* script. Parameters are also passed to *utmlp* in the call in addition to the data to be printed. *utmlp* then passes the data by default to the lp command, see information on the *utmlp* script in section "PTERM - define the properties of a client/printer and assign an LTERM partner").

(PRINTER, printertype,[class]) (only on Unix and Linux systems)

Printer with extended parameters.
To output the data, the printer process (*utmprint*) calls the *utmlp* script. Parameters are also passed to *utmlp* in the call in addition to the data to be printed. *utmlp* passes the data with the value of *class* set to *destination* to the lp command (for information on the *utmlp* script *utmlp* see below).

printertype
Designates the printer type of the printer to be used for printing. If there are special characters in the value of the *printertype* parameter, then you must place the value in single quotes.
Maximum length of *printertype*: 8 characters

class
Name of the printer group (printer class).
If the name of the printer group contains special characters, then you must place the value in single quotes.
Maximum length: 40 characters
Default value: value of *ptermname*

Information on the utmlp script:

The *utmlp* script is also supplied with openUTM. You will find it in the $UTMPATH/shsc directory.

The parameters which are transferred to the script *utmlp* dependent on the PTERM statement are documented in the script itself.

The script is accessed at runtime using the $PATH variable.

You can edit the script to modify the message before printing or print it over the network, for example.

If printing was successful, the script returns exit code 0 (null). If the script returns an exit code other than 0, then the connection to the printer process is cleared and another attempt is made to output the data once the connection has be reestablished.

> **i** With clients of type APPLI,SOCKET or UPIC-R, it may appear to openUTM that a connection to the client still exists, even though the client is no longer actually linked to the application and therefore attempts to reestablish the connection. For this purpose, the client sends a connection request to openUTM, which causes openUTM to shut down the “existing” connection.
>
> With clients of type APPLI or SOCKET, openUTM then automatically initiates the setup of a new connection.For UPIC clients, the initiation to establish a new connection must be made by the UPIC client.

STATUS= Status (locked or unlocked) of the client/printer when the application is started.

ON — The client or printer is unlocked. If the LTERM partner used by the client/printer to sign on to the UTM application is not locked, connections can be establish or may already be in place.

Default: ON

OFF — The client or printer is locked. Connections cannot be established between the client/printer and the local application. The client/printer can be released by the administrator.

TERMN= termn_id

Identifier up to two characters in length, which indicates the type of client. openUTM provides this identifier to the application program in the KCTERMN field of the KB header. *termn_id* is not queried by openUTM, but can be used by the user for analysis purposes.

Default values:

If this operand is not specified, openUTM sets the KCTERMN field to the default ID of the partner type specified in the PTYPE operand. However, the user can select other values if desired.

BS2000 systems:
The default values are listed in the partner type table for the PTYPE= operand in section "PTERM - define the properties of a client/printer and assign an LTERM partner".
If TERMN is not explicitly specified for clients generated with PTYPE=*ANY, openUTM does not enter the terminal mnemonic in KCTERMN until the connection is established. This is the default terminal mnemonic of the type specified in the user services protocol of the connection request.

Unix, Linux and Windows systems:
The default values are listed in the table below.

T-PROT= This operand is only supported on Unix, Linux and Windows systems.

The address format with which the OSI TP partner signs on to the transport system. Is only relevant for PTYPE=SOCKET, APPLI and UPIC-R.

Information on the following address formats can be found in the section "PCMX documentation" (openUTM documentation).

RFC1006 — Address format RFC1006

SOCKET — Communication is conducted via the socket interface.
SOCKET may only be specified if the name of the local UTM application that you specified in BCAMAPPL is generated with T-PROT=SOCKET.
The specification of a port number in the LISTENER-PORT operand is mandatory.

The default value for T-PROT depends on the PTYPE specification:

T-PROT=RFC1006 when PTYPE=APPLI or UPIC-R
T-PROT=SOCKET when PTYPE=SOCKET

TSEL-FORMAT=

This operand is only supported on Unix, Linux and Windows systems.

The format identifier of the T-selector in the transport address of the client.
TSEL-FORMAT is only relevant for PTYPE=SOCKET, APPLI and UPIC-R.

The format identifier specifies the coding of the T-selector in the transport protocol. You will find more information in the section "PCMX documentation" (openUTM documentation).

T — TRANSDATA format

E — EBCDIC format

A — ASCII format

It is recommended to explicitly specify a value for TSEL-FORMAT for operation via RFC1006.

USAGE=

This specifies whether the communication partner is a dialog partner or purely an output medium.

D — The client is a dialog partner. Messages can be exchanged between the client and the local application in both directions.

UPIC clients (PTYPE=UPIC-R) are always dialog partners.

An LTERM partner with USAGE=D must not be assigned to a client with USAGE=O.

This is the default value if PTYPE=SOCKET, APPLI, UPIC-R, and for terminals.

O — The communication partner is a printer. Messages can only be sent from the application to the printer.

Default:
This is the only value permitted for partners generated with PTYPE=*RSO.

USP-HDR=

This parameter is used to control the output messages for which openUTM is to establish a UTM socket protocol header on this connection.
A description of the USP header can be found in the openUTM manual „Programming Applications with KDCS”.

This parameter is only relevant for PTERMs with PTYPE=SOCKET.

NO — openUTM does not create a UTM socket protocol header for any of the output messages.

Default.

MSG — Only when outputting K messages does openUTM create and prefix the message with a UTM socket protocol header.

ALL — openUTM creates and prefixes all output messages (dialog, asynchronous, K messages) with a UTM socket protocol header.

6.5.38 QUEUE - reserve table entries for temporary messages queues

The QUEUE control statement allows you to specify the number of temporary queues that are permitted to exist in the application at any one time. In the KDCFILE the appropriate number of table entries are reserved for temporary queues. You can also define the default settings for these queues.

Temporary queues are suitable, for example, for communication between two services. These can be created and deleted dynamically during operation using the KDCS calls QCRE and QREL.

The QUEUE statement may only be specified once during a generation run!

For more information about queues and possible applications please refer the openUTM manual “Concepts und Functions”.

QUEUE	NUMBER=queue-number [,QLEV=queue_level_number] [,QMODE = { STD | WRAP-AROUND }]

NUMBER= queue-number

Specifies the maximum number of temporary queues that are permitted to exist at any one time during an application run.

Minimum value: 1
Maximum value: 500.000

QLEV= queue_level_number

(**Q**ueue **Lev**el)
Specifies the standard value for the maximum number of messages that may exist at any one time in a temporary message queue.
The maximum number of messages can be defined specifically using the KDCS call QCRE (KCLA parameter) for each queue when the queue is generated. The default value generated with QLEV= is used if the value 0 is entered in the parameter KCLA.

QLEV=32767 means that the number of messages in the queue is not limited by default.

Default: 32767 (or in other words, an unrestricted queue length)
Minimum value: 1
Maximum value: 32767 (or in other words, an unrestricted queue length)

QMODE= (**Q**ueue **Mode**)
Determines the behavior of openUTM in the event that the maximum number of messages saved in a temporary queue has been exceeded and the queue level is thus reached.
The value generated here is used when dynamically creating a temporary queue if no other value is specified in the KDCS call QCRE.

STD openUTM rejects all additional messages for the queue with a negative return code if the queue level has been reached.

Default: STD

WRAP-AROUND openUTM continues to accept messages for the temporary queue, even if the queue level has already been reached. When writing a message to the queue openUTM deletes the oldest messages in the queue and replaces it with the new one.

6.5.39 REMARK - insert a comment line

The REMARK control statement allows you to insert a comment in the KDCDEF control statements. Comments must not extend beyond one line.

REMARK	`comment`

comment Any character string

A comment line can also be created by inserting an asterisk * in column 1.

6.5.40 RESERVE - reserve table locations for UTM objects

If you use the functions of the program interface KDCADMI for dynamic configuration during application operation, or want to add dynamic objects using the WinAdmin or WebAdmin administration workstation, then you must use the RESERVE statement to reserve table spaces in the object tables of openUTM at the KDCDEF generation.

Further information on dynamic configuration can be found in section "Changing theconfiguration of an application dynamically".

The RESERVE statement can only be issued once for each object type. The following is valid for a RESERVE statement with OBJECT=ALL:

- After RESERVE OBJECT=ALL is specified, it is not possible to enter any additional RESERVE statements.
- Before RESERVE OBJECT=ALL all object-specific RESERVE statements are permitted. These object-specific RESERVE statements take priority.

Due to internal dependencies, the KDCDEF generation tool may reserve more objects than specified in RESERVE statements. The exact number of objects of the reserved entries for an object type is output in a message.

```
RESERVE  OBJECT={ ALL [ ,CARDS=percent1[1] ] [ ,PRINCIPALS=percent2 [1] ]
         |
                  CON      |
                  KSET     |
                  LSES     |
                  LTAC     |
                  LTERM    |
                  PROGRAM  |
                  PTERM    |
                  TAC      |
                  USER [ ,CARDS=percent1[1] ] [ ,PRINCIPALS=percent2[1] ] }
         [ { ,NUMBER=number | ,PERCENT=percent3 } ]
```

[1]only permitted on BS2000 systems

OBJECT= Table locations are reserved for objects of the specified type. These objects can then be entered in the configuration dynamically as required.

ALL [,CARDS=percent1] [,PRINCIPALS=percent2]

(CARDS= and PRINCIPALS= are only permitted on BS2000 systems)
Table locations can be reserved for objects of type CON, KSET, LSES, LTAC, LTERM, PROGRAM, PTERM, TAC and USER, which are then entered dynamically.

With objects of type USER, CARDS=*percent1* means that up to *percent1*% of users entered dynamically can be defined with an ID card.
PRINCIPALS=*percent2* means that up to *percent2*% of users entered dynamically can be defined with a Kerberos authentication.

Default for *percent1*/*percent2*: 0%, i.e. no users can be entered dynamically with an identity card or Kerberos authentication.
Maximum value for *percent1*/*percent2*: 100%

CON — Table entries are reserved for the transport connections to LU6.1 partner applications, for example, for objects of type CON.

KSET — Table entries are reserved for the key sets, for example, for objects of type KSET.

LSES — Table entries are reserved for the LU6.1 session names, for example, for objects of type LSES.

LTAC — Table entries are reserved for the local service names via which the service programs in partner applications can be started. These are objects of the type LTAC.

LTERM — Table locations are reserved for objects of type LTERM

Please note that the following object components must be generated statically and cannot be entered dynamically:

- A client with PTYPE=APPLI cannot be assigned dynamically to an LTERM partner without an autosign USER with the name of a statically generated USER.
- A client with PTYPE=APPLI cannot be assigned dynamically to an LTERM partner without an autosign USER defined with the name of a statically generated user.
- the format handling system when using formats
- the sign-on procedure with #formats.

PROGRAM — Table locations are reserved for objects of type PROGRAM
Objects of type PROGRAM can only be entered dynamically in applications generated with load modules (BS2000 systems), shared objects (Unix and Linux systems) or DLLs (Windows systems).

Please note that the following object components must be generated statically and cannot be entered dynamically:

- Programming languages (PROGRAM ...,COMP=) must be generated statically using the PROGRAM statement.
- With ILCS-compatible languages (COMP=ILCS), the static generation of an ILCS program is sufficient.
- LOAD-MODULEs in PROGRAM must be generated statically using the LOAD-MODULE statement.
- For applications generated without load modules on BS2000 systems, the PROGRAM names specified when entering a new TAC must be generated statically.
- For applications generated without shared objects/DLLs, the program names specified when entering a TAC must be generated statically.

PTERM — Table locations are reserved for objects of type PTERM.

For each client with PTYPE=APPLI, SOCKET, UPIC-R or UPIC-L, openUTM implicitly creates a USER. If such clients generated dynamically, this must be taken into consideration in the NUMBER= or PERCENT= operand for OBJECT=USER.

Please note that BCAMAPPL names must be generated statically and cannot be entered dynamically.

TAC

Table locations are reserved for objects of type TAC.

Please note that the following object components must be generated statically and cannot be entered dynamically:

- TAC classes
- If TACs are to be created dynamically for X/Open program units, at least one X/Open TAC must be generated statically.

USER [,CARDS=percent1] [,PRINCIPALS=percent2]

(CARDS= and PRINCIPALS= are only permitted on BS2000 systems) Table locations are reserved for objects of type USER.

If user IDs have not been generated for an application, i.e. the generation does not contain any USER statements, table locations cannot be reserved for objects of type USER. This is because KDCDEF already reserves an

object of type USER internally for each reserved object of type LTERM. The number of users reserved by KDCDEF in this way is output in a UTM message.

BS2000 systems:
On BS2000 systems CARDS=*percent1* means that up to *percent1*% of users entered dynamically can be defined with an ID card. PRINCIPALS=*percent2* means that up to *percent2* % of users entered dynamically can be defined with a Kerberos authentication.

Default for *percent1*/*percent2*: 0%, i.e. no users can be entered dynamically with an identity card or Kerberos authentication. Maximum value for *percent1*/*percent2*: 100%

Please note that the following object components must be generated statically and cannot be entered dynamically:

- the format handling system when using formats
- the sign-on procedure with #-formats

UTM creates an internal user ID for all TS applications (PTYPE=APPLI/ SOCKET) and UPIC clients (PTYPE=UPIC-R). The NUMBER or PERCENT specification must be increased appropriately if these PTERMs are to be entered dynamically.

NUMBER= number

Maximum number of objects of the specified type which can be entered dynamically.

If OBJECTS=ALL, up to *number* objects of the types listed in the syntax diagram can be entered dynamically.

- NUMBER=0 The number of objects of the specified type can be increased dynamically to the maximum value, i.e. the maximum number of names that can be generated as specified in section "Number of names ".
- NUMBER!=0 This reduces the storage space occupied by the UTM application. If the number of objects to be reserved is greater than the maximum number of names that can be generated (see "Number of names "), then this statement has the same effect as NUMBER=0.

Minimum value: 0
Maximum value:

- 32 000 for LTAC, KSET, TAC and PROGRAM
- 65 000 for CON, LSES, LTERM, PTERM
- 500 000 for USER, LTERM, PTERM

The following also apply:

- The sum of the reserved entries for an object type and the number of statically generated names of the associated name classes must not exceed the maximum number of permitted entries for these name classes (see "Number of names").
- The sum of the reserved CONs and PTERMs must not be greater than 500 000.
- The sum of the reserved LSES and USER must not be greater than 500 000.

At the end of the KDCDEF run, the number of entries reserved for each object type is output with message K502.

PERCENT= percent3

Number of objects of the specified type which can be entered dynamically, expressed as a percentage of the total number of objects of this type which have been generated statically.

The advantage of a percentage specification is that the number of objects that can be entered dynamically automatically increases at the same rate as the number of statically generated objects of the respective type in each generation (assuming the RESERVE statements are not modified).

PERCENT=*percent3* has the same effect as the equivalent NUMBER=*number*, i.e. PERCENT=0 has the same effect as NUMBER=0.

PERCENT!=0 reduces the storage space occupied by the UTM application. If the number of objects to be reserved is greater than the maximum number of names that can be generated (see "Number of names"), this statement has the same effect as PERCENT=0.

Default: 10
Minimum value: 0
Maximum value: Number of names that can be generated

6.5.41 RMXA - define a name for an XA database connection (Unix, Linux and Windows systems)

The XA interface standardized by X/Open allows you to link openUTM to any Resource Manager that supports this interface, e.g. the database system Oracle. openUTM supports this connection via the XA CAE interface (CAE specification).

A separate RMXA statement must be issued for each Resource Manager instance.

The start parameters for the Resource Manager generally determine the database to which openUTM is linked via the Resource Manager.

```
RMXA   XASWITCH=name
       [ ,USERID=username | C‘username‘ ]
       [ ,PASSWORD=C'password' ]
       [ ,DLLIMPORT={ YES | NO } ]
       [ ,XA-INST[-NAME]=inst-name ]
```

XASWITCH= name

Name of the *xa_switch_t* structure of the Resource Manager, which is made known to openUTM. The value entered for *name* is predefined in the respective Resource Manager. The name my be up to 54 characters in length. Further information can be found in section "Defining database linking".)

This is a mandatory operand.

USERID= username | C‘username‘

If a user name is to be passed to the database system in lowercase characters, then you must use the format C'username'.

> Alternatively, the user name can be transferred to the database system by means of start parameters.
>
> It is possible to modify the user name and/or the password by means of dynamic administration.

PASSWORD = C'password'

Specifies a password for the database system. The password can be up to 30 characters in length. openUTM passes the password to the database system if the placeholder *UTMPASS has been specified for the password in the Open string. In other words openUTM replaces '*UTMPASS' in the start parameter by the value generated here.

Alternatively, the password can be transferred to the database system by means of start parameters.

DLLIMPORT= Specifies how the *xa_switch_t* structure of the Resource Manager is to be addressed.

YES — Only permitted in openUTM on Windows systems.
The *xa_switch_t* structure is addressed with `dllimport`.
You must generate DLLIMPORT=YES to connect to Oracle on Windows systems.

NO — The *xa_switch_t* structure is addressed using `extern`.

Default: NO

XA-INST-NAME= This parameter is permitted only if TYPE=XA was specified.

inst-name is the local name for the XA instance which is 1 to 4 characters long.

If more than one RMXA statement is generated for an application, the XA-INST-NAME values in the applications must differ. The parameter can be omitted for applications with just one RMXA statement.

The string specified for XA-INST-NAME must be specified after the string ".RMXA" in the prefix for the start parameters for this database.

Example:

Specification in the RMXA statement:

```
RMXA....,XA-INST-NAME=DB1
```

Specification in the start parameters for this data base:

```
.RMXADB1 RM="rm-name",OS="open-string",...
```

Default: blanks

6.5.42 ROOT - define a name for the ROOT table source

The ROOT control statement must be specified when creating a ROOT table source. It can be omitted if only the KDCFILE is to be created, i.e.neither the operand GEN=ALL nor GEN=ROOTSRC is specified in the OPTION statement. The ROOT statement can only be issued once.

`ROOT`	`rootname`

rootname mandatory operand. You have to specify:

BS2000 systems:
CSECT name of the KDCROOT table to be incorporated (Assembler program).

When using the ROOT dynamic loading mechanism, this module is loaded during application startup from the library specified in the start parameter TABLIB=*libname*. If this is not the case, the module must be linked statically.

Unix, Linux and Windows systems:
Name part of the file containing the ROOT table source as a C/C++ program. *rootname* is an alphanumeric name up to eight characters in length. The fully qualified name is *filebase/rootname*`.c` (Unix and Linux systems) or *filebase\rootname.c* (Windows systems).

6.5.43 SATSEL - define SAT logging (BS2000 systems)

The SATSEL control statement allows you to define which events from which UTM event class are to be logged using SAT (preselection of the events to be logged). This involves specifying the event class to which the events belong, and then restricting the logging procedure within each event class by defining whether only successful results or only unsuccessful results are to be logged.

The SATSEL statement can be issued several times. If an event class is specified in several SATSEL statements, the values entered in the first statement determine the logging mode.

If SAT logging is to be activated when the application is started, this must be defined during generation (MAX ..., SAT=ON).

If MAX ...,SAT=OFF is generated, you can use SATSEL to define the events to be logged in the generation, even if SAT logging is deactivated. In this case, the SATSEL statements are not effective, but SAT logging is predefined. When required, SAT logging can then be activated during operation (KDCMSAT administration command).

The event logging mode can also be defined using the SATSEL operand in the USER (user specific) and TAC (TAC-specific) statements. If entries are made in various statements, the following applies:

- Logging is switched on as soon as it is activated in a statement. The logging mode (SUCC, FAIL, or BOTH) is unique.
- SAT logging can be activated in several statements (SATSEL, USER, and TAC statements). If different logging modes are specified in the various statements, openUTM creates a superset of logging modes by ORing the individual settings. However, there is one exception: if an event class is set to OFF in the SATSEL statement, logging is deactivated for this event class even if it is activated in the USER or TAC statement. Further information on the possible combinations of SAT logging conditions and their effect can be found in the openUTM manual “Using UTM Applications on BS2000 Systems”.
- Each event to be logged (apart from SIGN, CHANGE-PW) is assigned to a USER and TAC. Logging of an event can thus be activated using the SATSEL statement (activate logging for a particular event) or the SATSEL operand of the USER or TAC statement.

> i You can find further information about the SAT logging and about possible combinations of conditions of SAT-loggings and their result in the openUTM manual “Using UTM Applications on BS2000 Systems”.

SATSEL	{ BOTH | SUCC | FAIL | NONE | OFF } , EVENT=(event1, event2, ...)

BOTH	Both successful and unsuccessful events of the class specified in EVENT are logged.
SUCC	Only successful events of the class specified in EVENT are logged. SAT logging is also performed as defined in the SATSEL operand of the USER and TAC statements.
FAIL	Only unsuccessful events of the class specified in EVENT are logged. SAT logging is also performed as defined in the SATSEL operand of the USER and TAC statements.
NONE	Event-specific SAT logging is not performed. SAT logging takes place only if activated for a specific user and/or TAC.

OFF — None of the events of the class specified in EVENT are logged, even if SAT logging has been activated in the USER or TAC command. This allows you to exclude events from logging which are not relevant to security (e.g. access to TLS areas), and thus to restrict the quantity of log data.

EVENT=(event1, event2, ...)

Event classes to be logged. The following event classes can be selected:

- SIGN — Events that occur when the user signs on.
- CHANGE-PW — Events that occur when the user password is changed.
- START-PU — Events that occur when starting a program unit run, or when accepting a dialog or asynchronous job.
- END-PU — Events that indicate the end of the program unit run.
- GSSB — Events that indicate access to a global secondary storage area (GSSB).
- TLS — Events that indicate access to a terminal-specific long-term storage area (TLS).
- ULS — Events that indicate access to a user-specific long-term storage area (ULS).
- ADM-CMD — Events that affect the execution of an administration command issued by direct input or via a program interface.

6.5.44 SESCHA - define session characteristics for distributed processing based on LU6.1

The SESCHA control statement allows you to define session characteristics between the local application and the partner application. The set of session characteristics defined here is stored under a name, which can then be assigned to an LPAP partner using the SESCHA= operand of the LPAP statement (see "LPAP - define an LPAP partner for distributed processing based on LU6.1").

When generating LU6.1 connections, you must bear in mind the information in section "Distributed processing via the LU6.1 protocol".

SESCHA	sescha_name [,CONNECT={ **Y**ES | **NO**}] [,CONTWIN={ **Y**ES | **NO** }] [,DPN=destination_process_name] [,IDLETIME=pacing_count_time] [,PACCNT=pacing_count_number] ,PLU={ **Y**ES | **N**O } ***additional operand on Unix, Linux and Windows systems*** [,MAP={ USER | **SYS**TEM | SYS1 | SYS2 | SYS3 | SYS4 }]

sescha_name — Name under which the session characteristics are combined. This is specified for the SESCHA= operand of the LPAP statement in order to assign these session characteristics to a particular LPAP partner.

CONNECT= — This defines whether the local application is to establish the connection to the partner application during startup.

- NO — The connection to the partner application must be established using an administration command.

  Default: NO
- YES — The connection to the partner application is established when the local application is started. If unsuccessful, openUTM repeats its attempt to establish the connection at intervals defined in MAX ...,CONRTIME=.

  CONNECT=YES can be specified both in the local application and in the partner application. This means that the connection is established automatically as soon as both applications are available.

CONTWIN= — (**cont**ention **win**ner)
This defines whether the local application is the contention winner or the contention loser. The contention winner application is responsible for managing the session and controlling the reservation of sessions by jobs. You must specify CONTWIN=Y in one of the two participating applications and CONTWIN=N in the other.

- YES — The partner application is the contention winner.

NO The local application is the contention winner.

In both cases, jobs can be started by either application. If both applications simultaneously attempt to initiate a job, the session is reserved by the job issued by the contention winner.

The correct selection of this parameter is important for performance in communication between two applications: CONTWIN=Y must be specified in one of the applications, and CONTWIN=N in the other.

Default: If PLU=N, the local application is the contention loser; otherwise, it acts as the contention winner.

DPN= destination_process_name

Entity that processes asynchronous messages. This operand is significant only for links to IBM systems.

Default: 8 blanks

IDLETIME= idle_time

Number of seconds for which the idle state of a session is monitored. If the session is not reserved by a job within the period specified in IDLETIME=, openUTM shuts down the connection.

IDLETIME = 0 means that the idle state of the connection is not monitored.

Default value: 0
Minimum value: 60
Maximum value: 32767

If you specify a value that is greater than zero and smaller than the minimum value, KDCDEF replaces the value with the minimum value.

MAP= This operand is only supported on Unix, Linux and Windows systems.

Controls the code conversion (EBCDIC <-> ASCII) for user messages exchanged between partner applications (OSI-LPAP) in the abstract syntax UDT.

User messages are passed in the message area on the KDCS interface in the message handling calls (MPUT/FPUT/DPUT).
openUTM does not generally execute any message handling for formatted messages (KCMF contains a format identifier).

USER openUTM does not convert the user messages, i.e. the data in the KDCS message area is transferred to the partner application unchanged.

Default: USER

SYSTEM | SYS1 | SYS2 | SYS3 | SYS4

UTM converts the user messages based on the conversion tables provided for the code conversion (see section "Code conversion"), i.e.:

- Prior to sending, the code is converted from ASCII to EBCDIC.
- After receipt, the code is converted from EBCDIC to ASCII.

The parameter values SYSTEM and SYS1 are synonymous.

The prerequisite is that the message has been created using the abstract syntax of UDT (KCMF = blanks).

UTM assumes that the messages contain only printable characters.

PACCNT= pacing_count_number

Maximum number of message segments of a long message which can be received by the local application without issuing a response. If this value is too high, this may result in network congestion or loss of messages.

If PACCNT=0, pacing does not take place.

Default: 3
Minimum value: 0
Maximum value: 63

> **CAUTION!**
> If only short messages are to be exchanged with the partner application (less than 4000 byte) then pacing should be deactivated (PACCNT=0); this saves on overhead in communication with the partner. If data flow problems still occur, then either the default must be reset or the generation of the transport system must be modified accordingly.

PLU= Application that opens the session, i.e. the primary logical unit (PLU).

YES The partner application is the primary logical unit.

NO The local application is the primary logical unit.

PLU=Y must be specified for one of the applications, and PLU=N for the other.

6.5.45 SFUNC - define function keys

The SFUNC control statement allows you to assign

- transaction codes,
- KDCS return codes transferred to the program units,
- KDC commands, and
- the stacking function

to the function keys of terminals.

It should be issued once for each function key to be used.

A function key can be selected in UPIC clients and transferred to the UTM application.
If openUTM receives a function key from a UPIC client, then only the parameter RET is evaluated. If the parameter is not generated, openUTM returns the return code 19Z for the MGET call.

> i
> - On BS2000 system F and K keys assigned by SFUNC cannot be used by FHS-DE (see the "FHS User Guide").
> - On Unix, Linux and Windows systems function keys are only relevant for UPIC clients. Only the RET operand is evaluated.

SFUNC	functionkey { [,CMD={ KDCDISP | KDCFOR[1] | KDCLAST | KDCOFF | KDCOFF-BUT | KDCOUT | KDCSIGN[1] }] | [,TAC=tac1] {[,RET=xxZ] | [,STACK=tac2]} }

[1]only on BS2000 systems

functionkey
: Short name for the function key. The following values are possible:

  With F keys, the value of the F key and an input message are displayed.
  With K keys, a short message is output indicating the value of the K key.
  K14 is required for ID card readers (see the openUTM manual „Programming Applications with KDCS”).

  BS2000 systems:
  Supported values are K1 to K14 and F1 to F24

  Unix, Linux and Windows systems:
  Supported values are F1 to F20

CMD=
: Name of the KDC command to be assigned to this function key. If the CMD is specified, it is not possible to define any further operands.

TAC= tac1

Name of the transaction code to be assigned to the function key. This must be defined as a service TAC using the TAC statement (CALL=FIRST/BOTH). If the function key is pressed outside the service, this has the same effect as entering the transaction code. If the function key is pressed within the service and neither RET nor STACK is specified, the first MGET call in the next program unit of this service issues return code 19Z.

RET= xxZ

Return code contained in the KCRCCC field of the communication area following an MGET call if a particular function key is pressed during a service.

If the function key at the terminal is pressed at the beginning of a service and TAC=*tac1* is not set, openUTM responds by outputting the messaged K009 or by starting the BADTACS program unit. At the first MGET call, the BADTACS program unit receives the return code assigned to the function key in the field KCRCCC.

If the function key activated from the UPIC client at the beginning of a conversation, that service is started that belongs to the TAC (TP_NAME) set by the UPIC client. At the first MGET call, the program unit receives the return code assigned to the function key in the field KCRCCC.

The RET and STACK operands are mutually exclusive.

Value range: 20 <= xx <= 39.
The assignment is freely selectable.

STACK= tac2

Name of the transaction code to be assigned to the function key. This must be defined as a dialog service TAC using the TAC statement (TYPE=D and CALL=FIRST/BOTH). If the function key is pressed within the service, the current service is stacked and the service with the transaction code *tac2* is started. If the function key is pressed outside the service, transaction code *tac1* is started. If transaction code *tac1* is not specified, pressing the function key starts the service with the transaction code *tac2*.

The RET= and STACK= operands are mutually exclusive.

Alternative escape keys on BS2000 systems

For K and F keys not available on the keyboard the following alternative keys can be used:

Key	Alternative
K1	
K2	
K3	
K4	ESC V
K5	ESC W
K6	ESC M
K7	ESC N
K8	ESC O
K9	ESC ?
K10	ESC >
K11	ESC =
K12	ESC <
K13	ESC ;
K14	ESC :
F1	
F2	
F3	
F4	ESC ^
F5	ESC _

6.5.46 SHARED-OBJECT - define shared objects/DLLs (Unix, Linux and Windows systems)

The SHARED-OBJECT control statement allows you to define

- on Unix and Linux systems: the name and properties of a shared object if programs are to be exchanged using the dynamic linker.
- on Windows systems: the name and properties of DLLs used for dynamic loading.

The program exchange functions are supported on all Unix and Linux systems except AIX systems.

```
SHARED-OBJECT   shared_object_name
                [ ,DIRECTORY=directory_name ]
                [ ,LOAD-MODE={ STARTUP | ONCALL } ]
                [ ,VERSION=version ]
```

shared_object_name
: Name of the shared object/DLL up to 32 characters in length.

DIRECTORY=
: directory_name

  Directory in which the shared object is stored. *directory_name* can be up to 54 characters in length.

  Default: No entry, i.e. the current directory is used.

  Unix and Linux systems:
  If a directory is not specified for DIRECTORY=, the *filebase* directory is searched for the shared object. If the shared object cannot be found there, the environment variable LD_LIBRARY_PATH is used as the search path.

LOAD-MODE=
: Load mode of the shared object/DLL

  STARTUP
  : The shared object/DLL is loaded when the application is started.

    Default: STARTUP

  ONCALL
  : The shared object/DLL is loaded when the first call of a program unit or of a conversation exit is issued.

    Shared objects/DLLs generated with LOAD-MODE=ONCALL can only be exchanged if the openUTM version support for shared objects/DLLs is used.

VERSION= Shared object/DLL version up to 24 characters in length.

VERSION= is evaluated only if openUTM version support is used. This is described in the openUTM manual “Using UTM Applications on Unix, Linux and Windows Systems”.

Default: No version specification

> On Windows systems it is recommended that you use the VERSION= operand since the search for the "lexically largest name" can return unexpected results on Windows systems.

If VERSION= is not specified and if *shared_object_name* is a directory name, the shared object/DLL is addressed using the highest version name (in lexical terms). openUTM regards the version name merely as an identifier, i.e. the lexical sequence does not necessary mean “older” or “newer”. The UTM administrator is responsible for version management.

- Shared object file name **without** version support (Unix and Linux systems):
  The fully qualified file name of the shared object is *directory_name /shared_object_name.* If *directory_name*/*shared_object_name* is a directory the file is loaded with the highest file name (in lexical terms) from this directory.
  If the generation statement is `SHARED-OBJECT aaa.so, DIRECTORY=`. the file `./aaa.so` is loaded.
- Shared object file name **with** version support:
  Unix and Linux systems:
  The fully qualified file name of the shared object is *directory_name /shared_object_name/version.*
  If the generation statement is `SHARED-OBJECT aaa, DIRECTORY=., VERSION=V1.S0` the file `./aaa/V1.S0` is loaded.
  Windows systems:
  The fully qualified file name of the shared object is *directory_name**shared_object_name**version*.a
  For the generation specification,
  `SHARED-OBJECT aaa, DIRECTORY=., VERSION=V1` the file `.\aaa\V1.dll` is loaded.
  The suffix `.dll` is added automatically.

6.5.47 SIGNON - control the sign-on procedure

You can specify options and parameters for the sign-on procedure of your UTM application with the SIGNON control statement. The signing on of users is controlled by the SIGNON parameter.

The parameters UPIC, RESTRICTED and CONCURRENT-TERMINAL-SIGNON are only relevant if a sign-on service is generated.

If you enter an invalid value for the SIGNON operand, then KDCDEF uses the corresponding default value. This is currently done without outputting a corresponding message (see the following descriptions of the operands).

```
SIGNON   [ CONCURRENT-TERMINAL-SIGNON=%_value ]
         [ ,GRACE={ NO | YES } ]
         [ ,MULTI-SIGNON={ YES | NO } ]
         [ ,OMIT-UPIC-SIGNOFF={ YES | NO } ]
         [ ,PW-HISTORY=number ]
         [ ,RESTRICTED={ YES | NO } ]
         [ ,SILENT-ALARM=number1 ]
         [ ,UPIC={ YES | NO } ]
```

CONCURRENT-TERMINAL-SIGNON=%_value

This is only relevant when your application is generated with a sign-on service.
You specify the percentage of users generated for which a sign-on service may be active at the same time in CONCURRENT-TERMINAL-SIGNON. openUTM attempts to allocate the necessary resources according to this specification.

The value *%_value* is based only on sign-on services that are started for terminal users and TS applications.

Default: 25 (%)
Minimum value: 1 (%)
Maximum value: 100 (%)

If you enter a value < 1 or > 100 for *%_value*, KDCDEF sets the default value of 25 % without outputting a message.

GRACE=

(Grace-Sign-On)
Specifies if a user may still change his or her password after the password validity period has expired (see USER PROTECT-PW, "USER - define a user ID").

YES

The user can still change his or her password after the password validity period has expired.
The change must be made within the sign-on procedure, before the user is completely signed on.
If a sign-on service is activated, the password can be changed there using the KDCS call SIGN CP, regardless of the client type. A sign-on service is always activated when a user signs on via a connection for whose transport access point a sign-on service has been generated.

The table below shows how the individual client types behave when a password has expired and how this behavior depends on whether a sign-on service is activated.

Client type	**Behavior if the password has expired** [1)]
UPIC	Regardless of whether a sign-on service is activated, the password can be changed using the *Set_Conversation_Security_New_Password* function.
BS2000 terminal	If the password is blanked out, openUTM prompts the user to change the password, regardless of whether a sign-on service is activated.
	If the password is not blanked out, openUTM prompts the user to change the password only if no sign-on service is activated.
Terminal on Unix, Linux and Windows systems	openUTM prompts the user to change the password, regardless of whether a sign-on service is activated.
TS application	The user can no longer change the password without activation of a sign-on service.
HTTP client	The user can not change the password.

[1)] The password can always be changed via the administration interface. By default, passwords with limited periods of validity are immediately set to "expired" when changes are made via the administration interface. If you want to prevent this, then you must explicitly request this in the administration interface.

Note the following particularities after regeneration or change generation:

- If, after regeneration (followed by a KDCUPD run), the password of a user becomes invalid because the complexity requirement has been increased, the user can change his or her password in the sign-on service only (using SIGN CP).
- After regeneration (without a subsequent KDCUPD run), openUTM forces users to change passwords generated with a validity period when they first sign on.

NO

The user cannot change his or her password after the validity period has expired. The password may only be changed by an administrator after the validity period has expired.

Default: NO

MULTI-SIGNON=

Specifies if a user may be signed on to the application multiple times under the same user ID simultaneously.

> The MULTI-SIGNON operand does not have any effect on the receiving and starting of asynchronous services via OSI TP.

YES

The following cases can arise:

- The user ID is generated with USER..,RESTART=NO:
  In this case the user may sign on to the application a multiple number of times simultaneously. However, the user may only sign on once through an application. The user can also sign on via UPIC, APPLI, SOCKET and OSI TP connections.
- The user ID is generated with USER...,RESTART=YES:
  In this case the user may sign on no more than once to the application, although additional job-receiving services can be active in the application for the user if these services are started via OSI TP connections and the commit functional unit was selected.

NO

Every user ID may only be signed on once, and no more than one dialog service can be active at a time for each user.

Default: YES

OMIT-UPIC-SIGNOFF=

Specifies whether a user who has signed on over a UPIC connection remains signed on or not after the conversation has finished.

YES

If a user has signed on over a UPIC connection, they remain signed on after the conversation has finished. This user is only signed off

- if another user 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 no other user is passed in the UPIC protocol, no sign-on service is started before the UPIC conversation is started.

If the application is generated without users, the user ID is never changed for an existing connection. In this case, therefore, a sign-on service is only started where necessary before the first conversation is started after the connection has been established.

Default in UTM cluster applications.

NO

If a user has signed on over a UPIC connection, they are signed off after the conversation has finished.

Default in standalone applications.

PW-HISTORY=

number

Specifies if and how many password changes are to be maintained by openUTM in the password history.

If you enter a value > 0 for *number*, then openUTM maintains a password history. *number* is the number of passwords for a user ID that are recorded by openUTM.

If a user changes his or her password and if a maximum period of validity is generated for the password in the USER statement, then the new password must be different from the current password and the last *number* of passwords used by the user. number=0 means that openUTM will not maintain a password history.

Default: 0
Minimum value: 0
Maximum value: 10

If you specify a value > 10 for PW-HISTORY, then KDCDEF sets it to the maximum value of 10.

The password history only applies to the user; the administrator can change the password irrespective of the history.

RESTRICTED=

Specifies if DB calls and access to global UTM storage is prohibited in the first part of the sign-on service.

YES

DB calls and access to global UTM storage is prohibited in the first part of the sign-on service.

NO

DB calls and access to global UTM storage is permitted in the first part of the sign-on service.

Default: YES

SILENT-ALARM=

number1

Specifies the number of unsuccessful sign on attempts that may occur one after the other via an LTERM partner or a terminal user. A silent alarm (message K094) is triggered when this number is exceeded. The message is output after *number1* unsuccessful sign-on attempts in a row by a user or by a client.

Default: 10
Minimum value: 1
Maximum value: 100

UPIC=

This is only relevant when a sign-on service is generated in your application.
With UPIC= you specify in UPIC if the sign-on service is activated when a UPIC client wants to start a conversation.

YES	If a sign-on service is generated for the transport system end point (BCAMAPPL) via which the UPIC client has connected to the application, this is started before every UPIC conversation.
NO	No sign-on service is started for UPIC clients. Default: NO

6.5.48 SUBNET - define IP subnets

For UTM applications IP subnets can be generated.

The generation of subnets allows

- to restrict access to the UTM-application to communication partners from a particular IP address range
- to suppress name resolution via DNS for communication partners from a particular IP address range. IP addresses from an address range thus defined can consequently be assigned a permanent name (so-called "mapped name"). This assignment is only effective for an external connection request.

Multiple IP subnets can be defined for a UTM application. In the case of an external connection request they are evaluated in the order in which they are defined in the KDCDEF input. IPv4 addresses are compared only with IPv4 subnet addresses and IPv6 addresses only with IPv6 subnet addresses.

SUBNET	mapped_name [,BCAMAPPL=local_appliname] { ,IPV4-SUBNET=X'ipv4_addr' | IPV6-SUBNET=X'ipv6_addr' } [,RELEVANT-BITS=number] [,RESOLVE-NAMES=YES | NO]

mapped_name

Name for the subnet, up to 8 characters long.
The name specified as *mapped_name* must begin with the character "*" (asterisk) and be defined as PRONAM in a TPOOL statement.
The string "*ANY" may not, however, be specified as *mapped_name*, i.e. subnets are not supported in conjunction with TPOOL PRONAM=*ANY.
The *mapped_name* must be unique, i.e. the same *mapped_name* may not be specified in more than one SUBNET statement.

BCAMAPPL= local_appliname

Name of a local UTM application as defined with MAX ...,APPLINAME= or in a BCAMAPPL statement.
local_appliname must be specified in BCAMAPPL= in the TPOOL statement, i.e. the TPOOL is assigned to the subnet via the name pair specified in the SUBNET statement comprising *mapped_name* and *local_appliname*.

Default: name, specified under MAX...,APPLINAME=.

Only connections which are set up to the application name defined with BCAMAPPL are assigned to the TPOOL which is allocated to the SUBNET. In this case no name resolution takes place by means of DNS.

Connections which come from the same subnet but are set up using a different application name are treated like normal connections. In other words, for these connections the host name is resolved by means of DNS, and the host name thus ascertained is used for assignment to the generated partner.

This enables, for instance, both UPIC partners and also other partners from the same subnet to sign on to a UTM application:

- At connection setup the UPIC partners specify the BCAMAPPL name which was defined in the SUBNET statement, and are assigned to the TPOOL.
- Other partners from this subnet, such as LU6.1 partners, specify a different local application name for connection setup.

IPV4-SUBNET=
IPV6-SUBNET= X‘ipv4_addr‘ or X‘ipv6_addr‘ respectively

IPv4 or IPv6 subnet address from whose range connections are to be mapped to the *mapped_name*.

The address is specified as a hexadecimal string.
Either IPV4-SUBNET or IPV6-SUBNET must be specified. The simultaneous specification of IPV4-SUBNET and IPV6-SUBNET is not permitted.

RELEVANT-BITS= number

Number of relevant bits for the subnet address (subnet mask). If an IP address which is to be checked has the same number of relevant bits in the subnet mask as a defined subnet mask, the IP address is mapped to the *mapped_name*.

Possible values:
IPv4 subnets: 1 ... 32
IPv6 subnets: 1 ... 128

Default:
IPv4 subnets: 24
IPv6 subnets: 64

RESOLVE-NAMES

The RESOLVE-NAMES parameter can be used to specify whether a DNS name resolution is to be used for connections that are established from this subnet.

If name resolution is used, the real processor name of the communication partner is displayed via the administration interface and in messages. Otherwise, the name of the subnet is displayed instead of the processor name.

Default on BS2000 systems: YES
Default on Unix, Linux and Windows systems: NO

6.5.49 TAC - define the properties of transaction codes and TAC queues

The TAC control statement allows you to define the name and properties of a transaction code and (permanent) message queues of the UTM application.

- **Transaction codes** are the “calling names” for program units of the application. You must always assign a program name (PROGRAM= operand) to a transaction code.
- **TAC queues** are application-wide message queues that exist independently of a program unit. The operand PROGRAM= may not be specified. TAC queues are service-controlled, which means that the program units of the UTM application are responsible for reading messages from queues, openUTM - unlike transaction codes - does not carry out scheduling.
- The **dead letter queue** is a TAC queue with the fixed name KDCDLETQ. It is always available for backing up queued messages sent to transaction codes or TAC queues which absolutely could not be processed, i.e. the maximum number of redelivery attempts may have been exceeded. In addition, it is possible to save asynchronous messages to LPAP or OSI-LPAP partners that could not be sent due to a permanent error and that are deleted from their message queue.

  The messages of the dead letter queue can be read with DGET BF/BN and moved for further processing to other message queues with DADM MV/MA. When moving, it must be ensured that the new target is of the same type (asynchronous TAC/TAC queue, LPAP partner or OSI-LPAP partner). Details can be found in the openUTM manual „Programming Applications with KDCS” for the KDCS call DADM.

  You cannot generate or process messages for the dead letter queue KDCDLETQ.

  The backing up of asynchronous messages in the dead letter queue can be enabled and disabled for each message destination individually using the DEAD-LETTER-Q parameter. This parameter is available in the following statements:

  - in the TAC statement for messages to asynchronous TAC or TAC queues
  - in the LPAP statement for asynchronous messages to LPAP partners
  - in the OSI-LPAP statement for asynchronous messages to OSI-LPAP partners

  Main jobs to message complexes with negative acknowledgement jobs are never backed up in the dead letter queue.

  The name KDCDLETQ is created for the dead letter queue during generation. The following properties are set for the generation:

  TYPE=Q, STATUS=ON, ADMIN=N, QMODE=STD, QLEV=32767

  The properties of this TAC queue can also be defined in a separate TAC statement.

A message to a TAC queue cannot be processed when the transaction containing the DGET FT/NT or PF/PN call is rolled back. A message to a asynchronous TAC cannot be processed when the asynchronous service started with PEND ER/FR terminates abnormally before reaching a synchronization point first.

Generating transaction codes

- The parameters QMODE, Q-READ-ACL and Q-WRITE-ACL have no significance for transaction codes.
- When defining transaction codes for program units containing calls of the X/Open CPI-C or XATMI interface, you must use the API= operand to assign the identifier of the program interface used to the TAC.
- The administration commands used to manage the application must also be defined as TACs. They can be generated as dialog TACs or asynchronous TACs. At least one administration TAC (preferably the KDCSHUT administration command) must be generated and defined in the application. You must also generate at least one user with administration authorization.

- The event services BADTACS, MSGTAC are defined by entering TAC statements with the privileged TAC names KDCBADTC and KDCMSGTC in the generation.
- An event service SIGNON (= sign-on service) may be defined in several ways:
  - using the privileged TAC name KDCSGNTC. You use this to define the event service for the access point specified in MAX APPLINAME=*appliname*. This event service is then also the default for all other access points that are generated using a BCAMAPPL statement.
  - using BCAMAPPL *appliname2*,SIGNON-TAC=*signon-tac* in conjunction with TAC *signon-tac.* You use this to define an own event service for the access point *appliname2*. In this way you can define several SIGNON services.

  The event service generated with KDCSGNTC is default for all other access points that are generated with a BCAMAPPL statement.
- For the event services BADTACS, MSGTAC and SIGNON, there are preset values for some operands. These are listed in the table below. These preset values cannot be modified for KDCBADTC, KDCMSGTC and KDCSGNTC. With TAC *signon-tac* that you must set the values as described below.

Operand in TAC statement	**Preset value for**		
	KDCBADTC	**KDCMSGTC**	**KDCSGNTC or TAC signon-tac**
ACCESS-LIST=	Blank	Blank	Blank
ADMIN=	NO	(freely selectable)	NO
API=	KDCS	KDCS	KDCS
CALL=	FIRST	FIRST	BOTH
ENCRYPTION-LEVEL=	NONE	NONE	NONE
LOCK=	0	0	0
SATADM= (BS2000 systems)	NO	NO	NO
SATSEL= (BS2000 systems)	NONE	NONE	NONE
STATUS=	OFF	OFF	OFF
TACCLASS=	no TAC class	16	no TAC class
TYPE=	D	A	D

  These default settings mean that, for example, the TACs KDCSGNTC, KDCBADTC and KDCMSGTC are not subject to access protection by key sets and lock codes and cannot be used by the user or specified in a FPUT or DPUT call.

  The TACs KDCBADTC, KDCSGNTC and KDCMSGTC are not subject to processing control by the TAC classes. This also applies for KDCMSGTC although KDCMSGTC is assigned to TAC class 16.

  All TACs running within a sign-on service are not subject to processing control by TAC classes.
- DEAD-LETTER-Q=NO is set for KDCMSGTC and cannot be changed.

- Note the following when generating TACs:
  - The programs assigned to the TACs KDCBADTC, KDCMSGTC and KDCSGNTC and TAC *signon-tac* must not be assigned to a load module to be loaded dynamically when the first call of one of its program units is issued (LOAD-MODULE statement with LOAD-MODE=ONCALL).
  - The event exit VORGANG and the program units of the service must be located in the same load module if the load module is generated with LOAD-MODE=ONCALL.
- UTM SAT administration commands (preselection commands) can only be generated as dialog TACs. The names of these TACs can be found in the openUTM manual “Using UTM Applications on BS2000 Systems”.

Generating TAC queues

Only the following operands of TAC statements are relevant for the generation of a TAC queue (TYPE=Q):

tacname, ADMIN, DEAD-LETTER-Q, QLEV, QMODE, Q-READ-ACL, Q-WRITE-ACL, STATUS and TYPE.

The ADMIN, QLEV, QMODE, Q-READ-ACL, and STATUS operands can be used as desired for the dead letter queue KDCDLETQ.

All other operands are not evaluated for TAC queues.

More information about TAC queues and the applications they make possible can be found in the openUTM manual “Concepts und Functions”.

TAC	tacname [,{ **AC**CESS-**L**IST=keysetname | LOCK=lockcode }] [,ADMIN={ YES | NO | READ }] [,API={ KDCS | (XOPEN,{ XATMI | CPIC })] [,CALL={ BOTH | FIRST | NEXT }] [, DEAD-LETTER-Q={ NO | YES } [,**E**NCRYPTION**-L**EVEL={ **N**ONE | 2 | 5[2] }] [,EXIT=conversation_exit] [,PGWT={ NO | YES }] *only allowed when TAC-PRIORITIES are used* [,PROGRAM=objectname] *only allowed with TYPE=D | A* [,QLEV=queue_level_number] [,QMODE = { STD | WRAP-AROUND }] [,Q-READ-ACL = keysetname] [,Q-WRITE-ACL = keysetname] [,STATUS={ ON | OFF | HALT | KEEP }] [,TACCLASS=tacclass] [,TACUNIT=tacunit] [,TYPE={ D | A | Q }] ***additional operands for BS2000 systems*** [,DBKEY=dbkey] [,RUNPRIO=priority] [,SATADM={ **N**O | **Y**ES }] [,SATSEL={ BOTH | SUCC | FAIL | NONE }] [,TCBENTRY=name_of_tcbentry_statement] [,TIME={ time1 | (time1,time2) }] ***additional operand for Unix, Linux and Windows systems*** [,RTIME=rtime]

[2] only on Unix, Linux, and Windows systems

tacname

Name of the transaction code or the message queue (TAC name) up to eight characters in length.

> The specified name must be unique and must not be assigned to any other object in name class 1. See also section “Uniqueness of names and addresses”.

ACCESS-LIST= keysetname

This allows you to define user access authorizations for this transaction code. ACCESS-LIST= may not be specified in conjunction with the operand LOCK=*lockcode*.

Under *keysetname* you must specify the name of a key set. The key set must be defined with a KSET statement.

A user is then only able to access the transaction code if the key set of the user (USER ..., KSET=), the key set of the LTERM partner via which the user is signed on and the key set specified under *keysetname* all contain at least once common key code.

If you specify neither ACCESS-LIST=*keysetname* nor LOCK=*lockcode* the transaction code is not protected and any user is able to call the transaction code.

Default: no key set

ADMIN= Authorization required by the user in order to call the transaction code, the TAC-queue or a service containing this transaction code as a follow-up TAC.

YES Meaning for one TAC (TYPE=A or D):
The TAC can only be called by the administrator or by a user with administration authorization. All functions of the program interface for administration can be used in the associated administration program.

Meaning for a TAC queue (TYPE=Q):
Only the administrator or a user with administration authorizations may read messages in this queue/write messages to this queue.

NO Administration authorization is not required for this TAC or this TAC-queue

READ Administration authorization is not required for this TAC or this TAC-queue

Only those functions of the program interface for administration that have read-only access to the application data can be used in the associated administration program (only KDCADMI with the operation code KC_GET_OBJECT).

API= Program interface used by the program unit belonging to the transaction code

This is a mandatory operand if you use the X/Open CPI-C or XATMI interface.

KDCS The program unit is a KDCS program.

Default: KDCS

(XOPEN,CPIC) The program unit is a CPI-C program.

(XOPEN,XATMI) The program unit is an XATMI program.

CALL= This specifies whether or not a service is started with the transaction code, i.e. whether the transaction code is the first TAC of a service or a follow-up TAC in a service.

BOTH The TAC can be used as the first TAC or a follow-up TAC in a service.

Default: BOTH

FIRST — The TAC can only be used as the first TAC in a service.

NEXT — The TAC can only be used as a follow-up TAC in a service. No queued jobs can be generated in this TAC.

> CPI-C programs must be generated with CALL=FIRST or BOTH.
> XATMI programs must be generated with CALL=FIRST.

DBKEY= dbkey

This operand is only supported on BS2000 systems.

This is only relevant if the program unit issues database calls.

dbkey is a name with a maximum length of 8 characters under which the activities of this transaction code are registered with the database system. The format of the key depends on the database system used. The DBKEY is only used for UDS databases and there serves as a special indicator for activity in the UDS monitor ("program-name"). In previous versions of openUTM, this name was also used in the context of permission checking (hence the designation DBKEY). For more information, see the chapter "BPRIVACY" in the UDS/SQL manual "Creation and Restructuring".

Default: UTM

The default value DBKEY=UTM causes the value of the start parameter DBKEY to be passed at the database interface (see openUTM manual “Using UTM Applications”, Start parameters).

DEAD-LETTER-Q= Specifies whether asynchronous messages of this message queue are to be placed in the dead letter queue after incorrect processing and unsuccessful redelivery.

The statement MAX ...,DEAD-LETTER-Q-ALARM can be used to enable monitoring the number of messages in the dead letter queue.

YES — Messages to this asynchronous TAC or this TAC queue which could not be processed are backed up in the dead letter queue if they are not redelivered and (with message complexes) no negative acknowledgement job has been defined from.

NO

Messages to this asynchronous TAC or this TAC queue which could not be processed are not saved in the dead letter queue.
This value must be generated for all dialog TACs, for asynchronous TACs with CALL=NEXT and for KDCMSGTC and KDCDLETQ.

Default: NO

> Main jobs to message complexes (MCOM) with negative acknowledgement jobs are never saved into the dead letter queue since the negative acknowledgement jobs are activated if an error occurs.
>
> If the number of messages in the dead letter queue is limited with QLEV, messages from asynchronous TACs or TAC queues can be lost if an error occurs. If this limit is not applied, the openUTM page pool must be dimensioned large enough. If there is a danger of a page pool bottleneck, the dead letter queue can be blocked during operation with STATUS=OFF.

ENCRYPTION-LEVEL=

In ENCRYPTION-LEVEL you set the minimum encryption level that must be used by a service started through this transaction code. The encryption level specified here applies to all messages that are send and received in the service.

NONE

Encryption of the messages is not necessary.
You must set ENCRYPTION-LEVEL=NONE for transaction codes generated with CALL=NEXT.

Default: NONE

2 | 5

A service can only be started with this transaction code if the input message from the client is transmitted in encrypted form.
Dialog output messages of the service are transmitted to the client in encrypted form.

The value specifies the algorithm to be used for encryption:
2: Encryption of input/output messages using the AES-CBC algorithm.
5: Encryption of input/output messages according to the AES-GCM algorithm.
Level 5 is currently only supported on Unix, Linux and Windows systems.

In relation to the encryption level of the connection (PTERM, TPOOL) this means:

- To call transaction codes generated with encryption level 2, the connection must have been established with at least encryption level 3.
- To call transaction codes generated with encryption level 5, the connection must have been established with at least encryption level 5.

If a client does not encrypt the first input message with at least the requisite encryption level or does not support encryption, then no service is started. The following exceptions apply:

- The calling client is generated as a trusted client) (PTERM/TPOOL ...,ENCRYPTION-LEVEL=TRUSTED).
- The service is an asynchronous service and is started locally.
- The service is started by means of service chaining.
- The service is started without user data.

If the transaction code is called without user data or started through service chaining, then the client must be able to encrypt because openUTM
transmits all dialog output messages in encrypted form and, for multi-step services, expects all additional input messages from non-trusted clients to be encrypted.

You may only specify ENCRYPTION-LEVEL= 2 | 5 for transaction codes used to start a service (CALL=FIRST or CALL=BOTH).

> **i** In applications for which no encryption functions are available t ransaction codes generated with ENCRYPTION-LEVEL=2 | 5 can only be started by trusted clients .

EXIT=

conversation_exit Name of the event exit VORGANG to be assigned to this TAC.

EXIT= can only be specified in conjunction with CALL=FIRST or CALL=BOTH. The event exit VORGANG must be defined in a separate PROGRAM statement.

Default: No event exit VORGANG

LOCK= lockcode

Lock code assigned to the transaction code of a service in the form of a numerical lock. *lockcode* is a number between 1 and the maximum value permitted by the application (MAX ...,KEYVALUE=).
This may not be specified in conjunction with the operand ACCESS-LIST=.

For data access control, key sets can be defined for (UTM) user IDs (USER) and for the LTERM/(OSI-)LPAP partners. If a service is secured by means of a lock code, it can only be started if the appropriate key code is contained both in the key set of the user ID, **and** in the key set of the LTERM/(OSI-)LPAP partner.

Services whose TACs are not secured with a lock code or an ACCESS-LIST can be called by any user ID and any LTERM/(OSI-)LPAP partner without restriction. Further information on the lock/key code and access list concepts can be found in the openUTM manual “Concepts und Functions”.

> **! CAUTION!**
> If the user and the LTERM/(OSI-)LPAP partner do not also have the key code for a continuation program called by this TAC, openUTM aborts the service with an error.

Default: 0 (the TAC is not secured with a lock code)
Maximum value: Value of MAX ...,KEYVALUE=*number*

PGWT You may only specify PGWT if the jobs to TAC classes are processed according to their priority in your application, i.e. the KDCDEF generation contains the TAC-PRIORITIES statement. You specify whether or not blocking calls (e.g. PGWT) are allowed to be executed in a program unit run that was started for this transaction code with PGWT.

YES Blocking calls are permitted.
If you specify PGWT=YES, then you must assign a TAC class to this transaction code, i.e. you must set TACCLASS= .

Note the following cases:

- CPI-C program units
  If a CPI-C program unit is to conduct dialog conversations in which send authorization is transferred to the conversation partner using a *Set_Send_Type* call with *send_type*= `CM_SEND_AND_PREP_TO_RECEIVE` or by issuing a Receive call in Send status, then the transaction code of this CPI-C program unit must be assigned to a TAC class generated with PGWT=YES.
- XATMI program units
  If an XATMI application contains both requests and conversational services, at least two tasks must be started and the transaction code for the service must be generated with PGWT=YES.

NO — Blocking calls are not permitted.

Default: NO

PROGRAM= objectname

Name of the program unit to which this TAC is to be assigned.

A program name must be generated for asynchronous and dialog TACs; the PROGRAM parameter is not permitted for TAC queues.

Default: Blanks, no program name

If the program is not loaded in application operation, or the access authorizations do not permit the call, openUTM calls the BADTACS dialog service. If BADTACS is not generated in the application, UTM outputs the message K009 instead.

QLEV= queue_level_number

(**q**ueue **lev**el)
For asynchronous transaction codes (TYPE=A), this operand specifies the maximum number of asynchronous messages that can be accommodated in the message queue of the transaction code. QLEV can be used to prevent the page pool from becoming overloaded with jobs for this TAC or this TAC queue. openUTM does not take the asynchronous jobs into consideration until the end of the transaction. It is possible to exceed the number of messages for a messages queue as specified in QLEV if several messages are created for the same queue in a transaction.

If an additional message is to be created once QLEV has been reached, the behavior of openUTM will depend on the setting made in QMODE= (see below).

Default: 32767
Minimum value: 0
Maximum value: 32767 (i.e. unlimited)
If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.

QMODE = (**Q**ueue **Mode**)
This determines the behavior of openUTM in the event that the maximum permitted number of messages saved in a queue has already been reached and thus the Queue Level has been reached.

STD — If, at the time of an FPUT or DPUT call, the number of messages saved in this queue is greater than or equal to the maximum number generated in QLEV=, the FPUT or DPUT call is rejected with 40Z or with an appropriate message, if this TAC was entered at a terminal.

WRAP-AROUND

Only for TACs with TYPE=Q (TAC queues):
openUTM continues to accept messages for this queue, even when the Queue Level has been reached. When writing the next messages to the queue openUTM deletes the oldest existing message from the queue providing that its start time has been reached and it is not currently being read.

Default: STD

Q-READ-ACL= read-keysetname

This parameter is only evaluated for TACs with TYPE=Q (TAC queues). This parameter is used to specify the authorizations that a user requires to be able to read and delete messages from this queue.

In this parameter you can specify the name if a KSET that is defined with a KSET statement. In this case, a user can only then have read access to this TAC queue if the key set (KSET) of the user and that of the logical terminal via which the user has signed on, both contain at least one key code that corresponds to the key code specified in the key set entered here.

If no key set is specified in Q-READ-ACL, all users are able to read and delete messages from this queue.

Default: no key set

Q-WRITE-ACL= write-keysetname

This parameter is only evaluated for TACs with TYPE=Q (TAC queues). It may not be specified for the dead letter queue.
This parameter is used to set the authorizations that a user requires to be able to write messages to this queue.

Using this parameter you can specify the name of a KSET that has been defined using a KSET statement. In this case, a user can only have write access to this TAC queue if the key set (KSET) of the user and that of the logical terminal via which the user is signed on, both contain at least one of the key codes contained in the key set specified here.

If no key set is specified in Q-WRITE-ACL, all users are able to write messages to this queue.

Default: no key set

RTIME= rtime

This operand is only supported on Unix, Linux and Windows systems.

Maximum real time (in seconds) available to a program unit started using this TAC. If the program unit runs over the specified time, openUTM terminates the service and outputs an error message (K017 with cause 70Z/XTnn, see the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”).

> **i** Monitoring of the program unit run also includes the PEND/PGWT call as well as any database calls. In the case of PGWT calls, the PGWT wait time is also included, i.e. in RTIME, you must also take account of the maximum wait time in PGWT (MAX PGWTTIME).

rtime = 0 means that the program unit real time is not monitored.

Default: 0
Minimum value: 0
Maximum value: 32767

RUNPRIO= priority

This operand is only supported on BS2000 systems.

BS2000 run priority of the TAC. This run priority is assigned to the UTM process in which the program unit runs (PROGRAM). You can thus use the BS2000 scheduling mechanisms to control the sequence of UTM program units. However, the RUNPRIO operand cannot influence the time at which openUTM starts a program unit.

When starting a program unit, openUTM attempts to set the run priority of the current process to the value defined in RUNPRIO for the current TAC. If the generated run priority is incompatible with the JOIN entries of the corresponding user ID, the run priority of the current process is not changed and openUTM outputs a corresponding K message. If the maximum permitted RUNPRIO values for the user ID and the job class are different, the value most beneficial to the user is permitted. If JOIN entries have not been defined, the run priority specified in RUNPRIO is set.

After the program unit is terminated, openUTM resets the run priority to its original value, unless it was changed during the program unit run using the CHANGE-TASK-PRIORITY command. In this case, the run priority set externally is retained after the end of the program unit.

If RUNPRIO=0, a TAC-specific run priority is not generated for this TAC.

Default: 0
Minimum value: 30 (highest priority)
Maximum value: 255 (lowest priority)

SATADM= This operand is only supported on BS2000 systems.

This operand defines whether UTM SAT administration authorization is required in order to call the TAC.

YES — The TAC can only be called by users/clients or partner applications for which administration authorization for SAT logging (PERMIT=SATADM) has been generated in the USER, LPAP or OSI-LPAP statement.

NO — The user/client or partner application does not require UTM SAT administration authorization to use the TAC.

SATSEL= — SAT logging mode when running the program unit called using this TAC.

If SAT logging is activated (MAX ...,SAT=ON), TAC-specific events are logged as defined in this operand during a program run under this TAC.

The SATSEL control statement is used to define the general SAT logging mode for all TACs and users. This can be supplemented by the SATSEL operand of the TAC statement, which allows you to define TAC-specific logging. If the logging of an event class is prohibited in the SATSEL statement, events of this class are not logged. (For information on the link between EVENT-, TAC- and USER-specific log settings, see openUTM manual “Using UTM Applications on BS2000 Systems”).

SATSEL can be generated even if SAT logging is deactivated (MAX ...,SAT=OFF). In this case, the statements are not effective when the application is started, but SAT logging is predefined. When required, SAT logging can then be activated during operation with the UTM SAT administration command KDCMSAT.

BOTH — Both successful **and** unsuccessful events are logged.

SUCC — Only successful events are logged.

FAIL — Only unsuccessful events are logged.

NONE — A TAC-specific SAT logging mode is not defined.

Default: NONE

STATUS= — Status (locked or unlocked) of the TAC or the TAC queue when the application is started.

ON — Meaning for TACs: The TAC is unlocked, and is available once the application is started, until such time as the administrator locks it.

Meaning for TAC queues: Read and write access is permitted for this queue.

Default: ON

OFF — The TAC is locked when the application is started. Jobs for this TAC are not accepted until the TAC is unlocked by the administrator.

If the transaction code belongs to a KDCS program unit and is generated with CALL=BOTH or CALL=NEXT, it is locked as a service TAC (first TAC of a service) but not as a follow-up TAC.

Meaning for TAC queues: The queue is locked to write access. Read access is permitted.

HALT

The TAC is locked in full when the application is started, i.e. even as a follow-up TAC in an asynchronous service or a dialog service.

If the TAC is called as a follow-up TAC, the service is terminated with PEND ER (74Z). The TAC must be released by the system administrator.
Asynchronous jobs already buffered in the message queue of the TAC are not started. They remain in the message queue until the TAC status is set to ON or OFF by the UTM administrator.

Meaning for TAC queues: The queue is locked to both read and write access.

KEEP

May only be specified for TAC queues and for asynchronous transaction codes that are also service TACs (CALL=BOTH or CALL=FIRST).
openUTM accepts jobs for the transaction code. The jobs are not processed, however, rather just written in the message queue of the transaction code. They are processed as soon as the administrator changes the status of the transaction code to ON or OFF.

You can use STATUS=KEEP to collect jobs that are to be executed later at a time when the application load is lower (e.g. at night).
To avoid overloading the page pool with too many temporarily stored jobs, you should limit the size of the job queue of the transaction code using the QLEV parameter.

Meaning for TAC queues: The queue is locked to read access. Write access is permitted.

> **i** The status us always set to ON for the KDCSHUT and KDCTAC administration commands, even if you specify a different value for STATUS. Your application can always be administered in this manner.

TACCLASS= tacclass

Assigns the transaction code a TAC class.

The TAC classes are required for controlling the processing of dialog and asynchronous jobs. Jobs that are assigned different TAC classes are started according to different criteria by openUTM. The TAC class, which is assigned a transaction code, controls whether a job is processed immediately or temporarily stored in the message queue of the transaction code first, and when it will be read out of the message queue and processed. There are two different methods available to control job
processing (see "Code conversion").

The following numerical values are permitted:

- 1 - 8 for dialog TACs
- 9 - 16 for asynchronous TACs

If asynchronous TAC classes are generated, then the value in MAX ...,ASYNTASKS must be greater than 0.

If your application is generated **without** TAC-PRIORITIES statements and encounters blocking calls in the program unit belonging to this TAC (e.g. the KDCS call PGWT), then you must specify the dialog or asynchronous TAC class for *tacclass* for which **TACCLASS PGWT=YES** is set.

If your application is generated **with** TAC-PRIORITIES statements, then you can assign any dialog or asynchronous TAC class to this TAC. You just need to set **TAC ..., PGWT=YES** in this case.

Default for dialog TACs:
Dialog TACs are normally not assigned to a TAC class. The program unit belonging to the dialog TAC is started as soon as a process retrieves the corresponding message from the job bourse of the application.

Default for asynchronous TACs:
The default value for asynchronous TACs is 16.

> **i** If the transaction code is generated with PGWT=YES, then you must assign a TAC class to the transaction code.

TACUNIT= tacunit

Specifies the number of accounting units that are charged in the accounting phase of the UTM accounting each time this transaction code is called. The accounting units are added to the accounting unit counter of the user ID that called the transaction code.
This operand is required only if openUTM is to collect accounting data (see also the ACCOUNT statement on "ACCOUNT - define the accounting functions" and "Accounting" in the openUTM manual “Using UTM Applications”. You must enter an integer here.

Default value: 1
Minimum value: 0
Maximum value: 4095

TCBENTRY= name_of_tcbentry_statement

Only relevant for transaction codes from program units that are generated with PROGRAM ...,COMP=COB1.

name_of_tcbentry_statement designates the name of a TCBENTRY statement in which the TCB entries assigned to this TAC have been combined.

Default: No name

TIME= This operand is only supported on BS2000 systems.

Supervise CPU consumption an elapsed run time for a program unit.

time1 Maximum CPU time (in milliseconds) available to the program unit with this TAC. If the program unit runs over the specified time, openUTM terminates the service and outputs UTM message K017 for dialog programs or K055 for asynchronous programs. KCRCCC is set to 70Z, and KCRCDC to XT20 (see the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”).
The value 0 means that the program unit started using this TAC is not subject to a timeout. Values 1 to 999 are not permitted and are replaced with 1000.

Default: 30000 ms
Minimum value: 0 ms
Maximum value: 86400000 ms

> **CAUTION!**
> With the administration TACs KDCSHUT, KDCSHUTA, KDCDIAG and KDCDIAGA, the value of TIME=*time1* should be set to a value greater than the default value (at least twice as large; >= 60000 ms).
>
> With KDCSHUT WARN, applications with large numbers of generated terminals may require more CPU time than permitted by the default value. (See also the openUTM manual “Administering Applications”.)The same is true when you request a diagnostics dump with KDCDIAG DUMP=YES in large applications.

time2 Maximum real time (in seconds) available to the program unit with this TAC.
If the program unit runs over the specified time, openUTM terminates the service with UTM message K017 for dialog programs or K055 for asynchronous programs. KCRCCC is set to 70Z, and KCRCDC to XTA0 (see the openUTM manual “Messages, Debugging and Diagnostics on BS2000 Systems”. The value 0 means that the real time is not monitored for the program unit started using this TAC.

> Monitoring of the program unit run also includes the PEND/PGWT call as well as any database calls. In the case of PGWT calls, the PGWT wait time is also included, i.e. in TIME=(...,*time2*), you must also take account of the maximum wait time in PGWT (MAX PGWTTIME).

Default: 0 s
Minimum value: 0 s
Maximum value: 32767 s

TYPE=	This defines whether jobs with this transaction code are processed in dialog, in an asynchronous mode or whether a TAC queue is created.
D	The TAC is a dialog transaction code, i.e. a job with this TAC is processed in the dialog with the job submitter. Default: D
A	The TAC is an asynchronous transaction code, i.e. a job with this TAC creates an asynchronous job in the message queue of the transaction code. Processing takes place independently of the job submitter.
Q	This TAC statement is used to generate a TAC queue. In a queue of this nature it is possible to use a FPUT or DPUT call to write a message to a queue and to use a DGET call to read a messages in the queue.

6.5.50 TACCLASS - define the number of processes for a TAC class

You specify the method used to control job processing in this UTM application with the TACCLASS control statement. This means that you specify the criteria used by openUTM to start the jobs for transaction codes that have been assigned a TAC class.

The specification of these criteria can also be done using the TACCLASS statement or the TAC-PRIORITIES statement.

A TAC class consists of a subset of the generated transaction codes of the application. These TACs are divided into TAC classes using the TACCLASS= operand of the TAC statement.

By generating at least one TACCLASS statement, you specify that job processing in your application is controlled by the limitation of the number of processes for the individual TAC classes. You may not issue any TAC-PRIORITIES statements in this case, then.

The TACCLASS statement allows you to define how many processes of the UTM application are allowed to work at the same time for the TACs of a TAC class. You can also specify in the PGWT operand whether or not blocking calls (e.g. the KDCS call PGWT) are allowed or not in program unit runs that are started by transaction codes of the TAC class. You may only assign the PGWT=YES property, i.e. blocking calls are allowed, to one dialog and one asynchronous TAC class.

The number of processes of a TAC class that you specify in the TACCLASS statement can be changed by the administrator. See the openUTM manual “Administering Applications” for details.

You can thus control the load on the UTM application exerted by the program units of individual TACs. For example, you can prevent long-running program units from blocking the application. If asynchronous services are used for distributed processing, then you can avoid situations where all the application processes available for asynchronous processing are allocated by this service.

Default values

All TAC classes are created implicitly in the KDCDEF generation if you generate a transaction code with the TAC ..., TACCLASS= statement, or if you generate a TAC class with TACCLASS.

If you do **not** issue any TAC-PRIORITIES statements, then you should write a TACCLASS statement for every TAC class used. In this case, openUTM assigns the minimum value for TASKS and TASKS-FREE to those TAC classes for which no TACCLASS statement is issued. You must always issue TACCLASS statements for TAC classes with PGWT=YES!

If you do not use TAC classes, i.e. the TACCLASS= operand is not specified in any TAC statement and there are no TACCLASS or TAC-PRIORITIES statements, then the following applies:

- Dialog TACs are processed without restriction.
- Asynchronous TACs are restricted by the number of processes specified in the ASYNTASKS start parameter. This value can be modified by the administration.

You will find a detailed description of the TAC classes and priority control in section "Job control - priorities and process limitations"

TACCLASSS	tacclass ,{ TASKS=number1 | TASKS-FREE=number2 } [,PGWT={ NO | YES }]

tacclass

Number of the TAC class for which the number of processes is to be specified. You may specify the following TAC classes:

- the dialog TAC classes 1 - 8
- the asynchronous TAC classes 9 - 16.

You assign a transaction code to this TAC class by specifying TACCLASS=*tacclass* in the corresponding TAC statement.

You may only specify an asynchronous TAC class if you have generated a non-zero value in MAX ...,ASYNTASKS.

Asynchronous transaction codes that are not assigned a TAC class are automatically assigned TAC class 16.

> The TAC class numbers are not priorities, rather only a designation for the TAC class. Only the number *number1* of permitted processes can determine the extent to which processing of a TAC class is restricted so that the TACs of other classes can be processed quicker. This is possible only if *number1* is less than the number of active processes of the application.

TASKS= number1

Maximum number of application processes that can be executed simultaneously for the TACs of this class. The values permitted here depend on the value of the PGWT operand and the values defined for the TASKS, TASKS-IN-PGWT, and ASYNTASKS operands of the MAX statement. The value ranges permitted for TASKS=*number1* are given in the table below.

	Class 1 - 8 Dialog TACs		**Class 9 - 16 Asynchronous TACs**	
Minimum value	PGWT=NO	PGWT=YES	PGWT=NO	PGWT=YES
	1	1	0	0
Maximum value	TASKS *)	TASKS-IN-PGWT *)	ASYNTASKS*)	The lesser of the two values: ASYNTASKS, *) TASKS-IN-PGWT *)

*) As specified in the MAX statement

This is a mandatory operand if TASKS-FREE= is not specified.

If you enter TASKS=0 for a dialog TAC class, openUTM automatically resets this value to 1.

> The total number of tasks entered in the TASKS= operand of the individual TACCLASS statements can be greater than the maximum number of processes permitted by the application (MAX ...,TASKS=).

TASKS-FREE= number2

TASKS-FREE specifies for

- dialog TAC classes: Minimum number of processes of the UTM application to be kept free for processing TACs of other classes.
- asynchronous TAC classes: Minimum number of processes permitted for asynchronous jobs (MAX ...,ASYNTASKS=) to be kept free for processing TACs of other classes.

Compared to the TASKS parameter, TASKS-FREE offers the advantage of dynamically adapting the number of processes permitted for a TAC class if the total number of application processes is changed.

The values permitted for TASKS-FREE=*number2* depends on the values defined for the TASKS and ASYNTASKS operands of the MAX statement.

The value ranges permitted for TASKS-FREE= are given in the table below:

	Class 1 - 8 Dialog TACs	**Class 9 - 16 Async. TACs**
Min. value	1	1
Max. value	TASKS-1 *)	ASYNTASKS *)
*) As specified in the MAX statement		

This is a mandatory operand if TASKS= is not specified.

If you enter TASKS-FREE=0, openUTM automatically resets this value to 1.

PGWT= (**p**ro**g**ram **w**ai**t**)

This specifies whether or not program units containing blocking calls (e.g. KDCS call PGWT) can be executed in this TAC class. (Further information on PGWT can be found in the openUTM manual „Programming Applications with KDCS” and in the openUTM manual “Concepts und Functions”).

YES

Blocking calls are permitted in this TAC class.
PGWT=YES can only be generated if MAX ...,TASKS-IN-PGWT !=0 is defined. It can be specified for up to one dialog TAC class and one asynchronous TAC class. Program units containing PGWT calls must be assigned to this TAC class.

- CPI-C program units
  If a CPI-C program unit is to conduct dialog conversations in which send authorization is transferred to the conversation partner using a Set_Send_Type call with send_type=CM_SEND_AND_PREP_TO_RECEIVE or by issuing a Receive call in Send status, the transaction code of this CPI-C program unit must be assigned to a TAC class generated with PGWT=YES, e.g.:

```
MAX TASKS=2
MAX TASKS-IN-PGWT=1
TACCLASS 1,TASKS=1,PGWT=YES
TAC CPIC1,PROGRAM=xyz,API=(XOPEN,CPIC),TACCLASS=1
```

- XATMI program units
  If an XATMI application contains both requests and conversational services, at least two tasks must be started and a TAC class that permits PGWT calls must be generated. A service is always linked to the task. This is not necessary for applications that only contain request/response services.

NO

Program units that contain blocking calls are not permitted in this TAC class.

Default: NO

> Blocking calls for asynchronous TACs are not processed until all jobs in the messages queues of dialog TAC classes have been executed.

Example

The table below shows how the value defined for TASKS-FREE affects that defined for the TAC class under certain marginal conditions.

- The CURRENT TASKS column contains the maximum number of processes currently available to the UTM application. CURRENT ASYNTASKS contains the maximum number of processes available for performing asynchronous services. The global maximum values for CURRENT TASKS and CURRENT ASYNTASKS are defined in the TASKS and ASYNTASKS operands of the MAX statement. During runtime, the current values can be modified dynamically within this upper limit using the TASKS and MAXASYN operands of the KDCAPPL command.
- The DIALOG column contains the maximum number of processes available for a particular DIALOG-TAC class if TASKS-FREE=*nn* is specified for this TAC class.
- The ASYNCH column contains the maximum number of processes available for a particular asynchronous TAC class if TASKS-FREE=*number2* is specified for this TAC class.

CURRENT TASKS	CURRENT ASYNTASKS	TASKS-FREE	DIALOG	ASYNCH
10	9	2	8	7
6	6	2	4	4
3	3	2	1	1
2	2	2	1	0
1	1	2	1	0
10	5	3	7	2
6	5	3	3	2

6.5.51 TAC-PRIORITIES - specify priorities of the TAC classes

With the TAC-PRIORITIES control statement you specify the method to be used to control job processing in this UTM application. This means that you specify the criteria used to start jobs for transaction codes that are assigned a TAC class.

You can also specify these criteria using the TAC-PRIORITIES statement or the TACCLASS statement.

A TAC class consists of a subset of the generated transaction codes of the application. The dividing of transaction codes into TAC classes is done in the TAC statement with the TACCLASS= operand.
If the TACCLASS operand is not specified, then dialog TACs are not assigned a TAC class and asynchronous TACs are not assigned asynchronous TAC class 16.

You can specify the following in particular with TAC-PRIORITIES:

- That the distribution of processes amongst the TAC classes is to be done according to priorities. You must not issue any TACCLASS statements in this case.
- The algorithm to be used to distribute the available processes of the application amongst the dialog and asynchronous TAC classes.
  Operands: DIAL-PRIO and ASYN-PRIO
- The maximum number of processes of the application that are allowed to process jobs to dialog TAC classes.
  Operand: FREE-DIAL-TASKS

Specifying priorities for the TAC classes

In priority control you can select between absolute, relative or equal priorities for dialog jobs and for asynchronous jobs. The control of job processing of dialog and asynchronous jobs is done separately from each other.

Jobs for dialog TACs that are not assigned any TAC class are processed regardless of the priorities set for dialog jobs. These jobs are always started immediately after they are received from the transport system.

The number of the TAC class plays a role for absolute and relative priorities. Jobs to TAC classes with a low number have a higher priority that jobs to TAC classes with a higher number. This means that for dialog TAC classes, TAC class 1 has the highest priority and TAC class 8 the lowest priority. For asynchronous TAC classes, TAC class 9 has the highest priority and TAC class 16 the lowest priority.

When absolute priorities are used, processes of the application that are free and available for processing the TAC classes are always assigned the TAC class with the highest priority, i.e. 1 or 9, as long as there are jobs waiting for this TAC class.

Only after there are no more jobs waiting in the TAC class with the highest priority are jobs waiting for the TAC class with the next lowest priority processed.

If you want to prevent jobs waiting for a TAC class with a lower priority from not being processed for a long time, then you should use relative priorities.

When relative priorities are used, jobs from TAC classes with higher priority are processed more often than jobs from TAC classes with lower priority.

When matching priorities are used, then the same number of jobs from each TAC class are processed as long as there are waiting jobs available.

Limiting the number of processes that process jobs to TAC classes

You can limit the number of processes that process jobs of a TAC class when using priority control for the TAC classes to keep some processes free for administrative tasks or internal jobs.

You limit the number of processes for the dialog TAC classes relative to the total number of processes using the FREE-DIAL-TASKS operand.
With MAX ASYNTASKS=(*atask_number*,...) you limit the number of processes for asynchronous TAC classes.

This limit is the same, however, for all asynchronous and for all dialog TAC classes.

Transaction codes that start program unit runs with blocking calls

When the TAC-PRIORITIES statement is used, transaction codes with blocking calls (e.g. the KDCS call PGWT) can be assigned any TAC class as long as the TASKS-IN-PGWT operand of the MAX statement is generated with a value > 0. You must generate TAC PGWT=YES for these transaction codes.

> If no TACCLASS statement or TAC-PRIORITIES statement is issued in the generation although the TACCLASS parameter was specified for at least one TAC statement, then the default values of the TACCLASS statement are applied. TAC priorities are not used in this case. See the TACCLASS description on "TACCLASS - define the number of processes for a TAC class" for more information.

You will find a detailed description of the TAC classes and priority control in chapter “Job control - priorities and process limitations”.

TAC-PRIORITIES	[DIAL-PRIO={ ABS | REL | EQ }] [,ASYN-PRIO={ ABS | REL | EQ }] [,FREE-DIAL-TASKS = number]

DIAL-PRIO = Specifies according to which priority free processes will be distributed amongst the dialog TAC classes with waiting jobs. Waiting dialog jobs can only arise when more jobs are obtained from the job bourse at a specific time than there are processes available for the dialog TAC classes. The jobs are then written to the job queues of the transaction codes from which they will then be read out and processed according to their priority by the processes that become free.

ABS
Absolute priority:
A free process is always assigned the TAC class with the highest priority (TAC class 1) as long as there are jobs waiting for this TAC class. TAC classes with lower priority are only serviced if there are no more jobs waiting in all TAC classes with higher priority.

REL
Relative priority:
Free processes are assigned TAC classes with higher priority more often than TAC classes with lower priority as long as there are jobs waiting for the TAC classes with higher priority. If jobs are available for all dialog TAC classes, then a free process will be assigned TAC class 1 twice as often as TAC class 2, and TAC class 2 will be assigned processes twice as often as TAC class 3, etc.

EQ
Equal priority:
As long as there are jobs available, all TAC classes are services equally often. This equal distribution can be interrupted if a TAC class does not have any jobs waiting for a while or when program unit runs with blocking calls (e.g. the KDCS call PGWT) often arise.

Default: EQ

ASYN-PRIO= Specifies according to which priority processes will be distributed amongst the asynchronous TAC classes with outstanding asynchronous jobs or interrupted asynchronous jobs.

If the maximum number of simultaneously open asynchronous services is reached (set in MAX ASYNTASKS=(...,*service_number*)), then no more asynchronous jobs are started. An interrupted open asynchronous service is selected according to its priority and resumed.

ABS Absolute priority:
A free process is always assigned the TAC class with the highest priority, i.e. TAC class 9, as long as there are asynchronous jobs or interrupted asynchronous jobs waiting for this TAC class. Free processes only process the jobs of a TAC class with a lower priority when there are no more outstanding or interrupted asynchronous jobs in the message queues of all TAC classes with higher priority.

REL Relative priority:
Free processes are assigned TAC classes with higher priority more often than TAC classes with lower priority as long as there are outstanding or interrupted jobs waiting for the TAC classes with higher priority. If jobs are available for all TAC classes, then a free process will be assigned TAC class 9 twice as often as TAC class 10, and TAC class 10 will be assigned processes twice as often as TAC class 11, etc.

EQ Equal priority:
As long as there are jobs available, all TAC classes are services equally often. This equal distribution can be interrupted if a TAC class does not have any jobs waiting for a while or when program unit runs with blocking calls (e.g. the KDCS call PGWT) often arise.

Default: EQ

FREE-DIAL-TASKS=number

With FREE-DIAL-TASKS you limit the total number of processes that may process jobs to dialog TAC classes relative to the number of all processes of the application. In *number* you specify the minimum number of processes of the application that are to be reserved for processing jobs that do not belong to any dialog TAC class.

> The maximum number of processes that may simultaneously process asynchronous jobs is not limited by FREE-DIAL-TASKS= . The MAX operand ASYNTASKS= *atask_number* is provided for this purpose.

Minimum value: 0 (no limit)
Maximum value: TASKS - 1 (TASKS from the MAX statement)
Default value: 1

Example
TASKS=7 and ASYNTASKS=2 was set in the MAX statement. FREE-DIAL-TASKS=3 is generated in the TAC-PRIORITIES statement. The application is operated with six processes. A maximum of three processes may process jobs in TAC classes 1 through 8 then. A maximum of two processes can process jobs in TAC classes 9 through 16. One process is reserved for dialog jobs that are not assigned a TAC class.

6.5.52 TCBENTRY - define a group of TCB entries (BS2000 systems)

The TBCENTRY control statement is permitted only for COB1 program units. COBOL program units that are not ILCS-compatible must be generated with PROGRAM ...,COMP=COB1 in openUTM. TCB entries offer benefits when used in conjunction with COBOL-DML, and in the case of a GOTO in a PERFORM routine. Further information can be found in the openUTM manual „Programming Applications with KDCS”.

TCB entries are used to create nested reentrant COBOL programs. They are required in the following cases:

- In conjunction with COBOL-DML:
  if the USE-DATABASE-EXEPTION clause is used in a DECLARATIVES subsection and the program run is terminated within these declaratives using PEND. In this case, you must specify the TCB entry I$ITCUPS; otherwise, the DECLARATIVES counter will not be reset, resulting in a COBOL error action. I$ITCUPS therefore resets the counter if a PEND occurs within the declaratives.
- In the case of a GOTO in a PERFORM routine:
  If a program unit is terminated in a PERFORM routine, the COBOL runtime system notes the return address. If you branch to the PERFORM routine using GOTO in the next program unit, the program unit behaves as if the PERFORM routine were still open and branches to the return address. Markers are reset by specifying a TCB entry (with any name).

TCB entries must also be made known to the COB1 compiler using the COBRUN parameter.

The TCBENTRY statement can be issued several times.

`TCBENTRY`	`tcbentry_groupname` `,ENTRY=( entry1,..., entry18 )`

tcbentry_groupname

Freely selectable name up to eight characters in length, which is used to address the group of TCB entries defined in this TCBENTRY statement, and to link the group of TCB entries to a TAC statement.

ENTRY= TCB entry name.

6.5.53 TLS - define a name for a TLS block

Each LTERM partner can be assigned a terminal-specific long-term storage area (TLS), which can contain several blocks. The TLS control statement allows you to define a name for a TLS block. The TLS block is then identified using the name of the LTERM partner (*ltermname*) and the block name defined here. openUTM provides each LTERM partner with a TLS block with this name. By issuing several TLS statements with different block names, you can define several blocks for each LTERM partner.

In the case of distributed processing, the TLS blocks defined in a TLS statement are also assigned to LPAP and OSI-LPAP partners.

You can issue up to 100 TLS statements.

`TLS`	`name`

name Name of a TLS block up to eight characters in length.

6.5.54 TPOOL - define an LTERM pool

The TPOOL control statement allows you to define the name and properties of an L**T**ERM **pool**. LTERM pools allow you to connect numerous clients with the same technical properties (partner and processor type) to a UTM application via LTERM partners. Printers are not supported in this case. The TPOOL statement merely defines the type (PTYPE=) and processor name (PRONAM=) for the client. The LTERM partner assigned to the client is specified dynamically in the UTM system code during connection setup on the basis of the LTERM partner name and client name defined in the TPOOL statement. This assignment applies only for the duration of a session, i.e. it is not a static assignment as in the case of the statement pair LTERM / PTERM. The clients contained in an LTERM pool need not be configured explicitly in the application (by defining a PTERM). The number of clients that can be simultaneously signed on is equal to the number of LTERM partners generated in the LTERM pool.

For clients that connect via an LTERM pool (i.e. that are not explicitly generated), the establishment of the connection can only be initiated "from outside", i.e. from the client itself. It is therefore not possible to establish a connection via UTM administration commands.

Also it is not possible to establish a connection via BCAM administration commands or using predefined BCAM connections on BS2000 systems.

The TPOOL statement allows you to define LTERM pools with different levels of availability for connection setup:

- With PRONAM=*processorname* and PTYPE=*partnertyp*, an LTERM pool is generated such that only clients of the same type located on the specified system can establish connections with the UTM application via this LTERM pool.
- With PRONAM=*ANY, all clients of a particular type can sign on to the UTM application irrespective of the system on which they are located.
- On BS2000 systems with PTYPE=*ANY, you can define an LTERM pool without specifying the client type. This LTERM pool can then be used by clients of all types located on the specified system to establish connections with the UTM application.
- With PRONAM=*ANY and PTYPE=*ANY, all clients on all systems can sign on to the UTM application on BS2000/PSD (open LTERM pool).

It is possible to define several LTERM pools, i.e. issue several TPOOL statements in each KDCDEF run. However, please note the following:

- *BS2000 systems:*

  The combination PRONAM / PTYPE / BCAMAPPL must be unique for LTERM pools for which the NEABT user protocol is defined (PROTOCOL=STATION). The combination PRONAM/BCAMAPPL must also be unique for LTERM pools with PROTOCOL=NO.

  The client must support the user services protocol specified in the TPOOL statement. PROTOCOL=NO must be generated for clients with PTYPE=APPLI, PTYPE=SOCKET or PTYPE=UPIC-R. PROTOCOL=STATION must be specified for LTERM pools generated with PTYPE=*ANY.

  During connection setup, openUTM takes the type (PTYPE) of the client generated with PTYPE=*ANY from the user services protocol (connection letter). openUTM then checks whether or not this client type is supported. If not, the connection request is rejected.

- *Unix, Linux and Windows systems:*

  The combination PRONAM/PTYPE/BCAMAPPL must be unique for LTERM pools.
  For LTERM pools, the maximum number of connections that can be established via one transport system endpoint at a time must be taken into account.

The LTERM partners of an LTERM pool are generated with LTERM ..., RESTART=NO. During connection setup, therefore, all messages buffered in the message queue of the LTERM partners of the LTERM pool are deleted. An LTERM-specific service restart is not performed. In applications generated without user IDs, a service restart cannot be executed after a connection is cleared and then re-established for clients that are connected to the application via an LTERM pool.

You can specify access rights for an LTERM pool (KSET operand) that clients connected through the LTERM pool may exercise. In applications with user IDs you can limit the access rights specified with KSET for LTERM pools generated for connecting UPIC clients or
TS applications using the USER-KSET operand. The access rights in KSET are then applicable to clients that explicitly specify a user ID when signing on. The limited access rights in USER-KSET take effect when the client does not specify a user ID when signing on, i.e. when the “connection user ID“ is active.

Using the LOCALE operand, you can define a client-specific language environment for each LTERM pool.

TPOOL	[,BCAMAPPL=local_appliname] [,CONNECT-MODE={ SINGLE | MULTI }] [,**E**NCRYPTION**-L**EVEL={ **N**ONE | 3 | 4 | 5[2] | **T**RUSTED }] [,IDLETIME=time] [,KSET=keysetname1] [,LOCK=lockcode] ,LTERM=ltermprefix [,MAP={ USER | SYSTEM | SYS1 | SYS2 | SYS3 | SYS4 }] ,NUMBER=number1 ,PRONAM [3] ={ processorname | C'processorname' | *ANY } ,PTYPE={ partnertyp | *ANY[1] } [,QLEV=queue_level_number] [,STATUS=({ ON | OFF }[, number2)] [,TERMN=termn_id] [,USER-KSET=keysetname2] [,USP-HDR={ALL | MSG | NO}] ***additional operands BS2000 systems*** [ANNOAMSG={ Y | N }] [,FORMAT= { + | * | # }formatname] [,KERBEROS-DIALOG={ **Y**ES | **NO** }] [,LOCALE=([lang_id][,[terr_id][,ccsname]])] [,NETPRIO={ **M**EDIUM | **L**OW } [,PROTOCOL={ N | STATION }]

[1] only on BS2000 systems

[2] only on Unix, Linux and Windows systems

[3] mandatory on BS2000 systems only

ANNOAMSG= (**anno**unc**e** **a**synchronous **m**e**s**sa**g**e)

This operand is only supported on BS2000 systems.

This applies only to LTERM pools used by terminals to sign on to the UTM application. It defines whether or not openUTM announces asynchronous messages before outputting them in the system line on the terminal.

Y	Asynchronous messages are announced in advance. The user must then request the message using the KDCOUT command. Default: Y
N	Asynchronous messages are sent without prior announcement.
BCAMAPPL=	local_appliname Name of the local UTM application. This name is then used to establish a connection between the client and the UTM application. *local_appliname* is defined either with MAX ..., APPLINAME= or in the BCAMAPPL statement (see "BCAMAPPL - define additional application names"). If you specify a value other than APPLI, SOCKET or UPIC-R for the PTYPE= operand, you can only specify the name defined with MAX ...,APPLINAME=*appliname* for *local_appliname*. Default: *appliname*, specified under MAX ...,APPLINAME=. *BS2000 systems:* The BCAMAPPL statement allows you to define whether or not NEA or ISO transport protocols or native TCP/IP (socket interface) are to be used when communicating with partners that sign on to this application. To establish a connection with the UTM application, the client must generally specify *local_appliname* as the partner name. One exception are LTERM pools that are generated with PTYPE=SOCKET. In this case, clients that connect via the LTERM pool must know the port number on which the UTM application "listens". This port number is specified in BCAMAPPL LISTENER-PORT= . *Unix, Linux and Windows systems:* The BCAMAPPL name specified in the CLUSTER statement is not permitted here.
CONNECT-MODE=	This defines whether a client can use the same name for multiple sign-ons to the UTM application via this LTERM pool.
SINGLE	Multiple sign-ons via the LTERM pool under the same name are not permitted. Default: SINGLE

MULTI

This is permitted only for LTERM pools that are used by UPIC partners or TS applications to connect.
A UPIC client program (PTYPE=UPIC-R or UPIC-L) or TS application (PTYPE=APPLI or SOCKET) can connect several times to the UTM application via the LTERM pool under the same name. A new name need not be created for each connection.

A UPIC client or TS application can connect a maximum of *number1* times to the LTERM pool (see NUMBER=*number1*, "TPOOL - define an LTERM pool").

In the case of CONNECT-MODE=MULTI, the UTM application does not identify the communication partner or the connection to the partner (as usual) using the name of the partner that the partner specified when the connection was established. The UTM application does not even know the partner under its application name. Instead, the partner is identified using the name of the pool LTERM partner (*ltermname*) through which it is connected. In order for openUTM to be able to uniquely identify the partner, the triplet consisting of the *ltermname* of the LTERM pool, the *processorname* and the *local_appliname* must not be explicitly generated in any PTERM, CON or OSI-CON statement. Additionally, the name that the partner specifies when establishing the connection may not match any LTERM name of the LTERM pool.

ENCRYPTION-LEVEL=

Only relevant for UPIC clients that support encryption and on BS2000 systems for some terminal emulations that support encryption also.

In ENCRYPTION-LEVEL you set the minimum encryption level for the communication with the clients, that connect via an LTERM pool with the application.

You specify whether or not the UTM application should request encryption of the message on the connection via the LTERM pool to the client. You can also define the client as a "trusted" client. This means that every client that connects via this LTERM pool is considered to be a trusted client.(see also section "Message encryption on connections to clients" for more information on encryption).

Default values:

TRUSTED is the default value for:

- HTTP clients und USP-Socket applications, which connect via a transport system access point (BCAMAPPL), that is configured with T-PROT=(..., SECURE).
- Local UPIC clients (PTYPE=UPIC-L) on Unix, Linux and Windows systems

Other values for these partners are changed to TRUSTED by KDCDEF without issuing a message.

NONE is the default value for

- all other types of communication partners.

For partners with PTYPE different from UPIC-R, and on BS2000 different from T9763, the values 3, 4, 5 are changed to NONE by KDCDEF without issuing a message.

You can specify the following:

NONE

Encryption of the messages exchanged between the client and the UTM application is **not** requested by openUTM.
However, passwords are always transmitted in encrypted form provided both partners support encryption.
Services for which encryption was generated for their service TACs (see ENCRYPTION-LEVEL in the TAC statement starting in section "TAC - define the properties of transaction codes and TAC queues") can only be started by this client if the client explicitly selects an encryption level that corresponds to at least the required level when establishing the conversation or connection.

Default: NONE

3 | 4 | 5

Messages exchanged between the client and the UTM application are encrypted by openUTM by default. The value specifies the encryption level. Only clients that support at least this encryption level can connect via this LTERM pool. If a client does not support the specified encryption level, openUTM rejects connection setup to the client.

The values have the following meaning:

3 Passwords and input/output messages are encrypted using the AES-CBC algorithm. An RSA key with a key length of 1024 bits is used to exchange the AES key.

4 Passwords and input/output messages are encrypted using the AES-CBC algorithm. An RSA key with a key length of 2048 bits is used to exchange the AES key.

5 Input/output messages are encrypted using the AES-GCM algorithm. The AES key is agreed using the Ephemeral Elliptic Curve Diffie-Hellman method (ECDHE). An RSA key with a key length of 2048 bits is used to sign the public Diffie-Hellman key of the server.
Level 5 is currently only supported by openUTM for LUW platforms.

BS2000 systems:
VTSU encryption is used for VTSU partners.

> i If the application is generated with OPTION GEN-RSA-KEYS=NO, no RSA keys are created in the KDCDEF run. In order to use the encryption functions, you must create the required keys using administration facilities (KC_ENCRYPT or WinAdmin or WebAdmin) or transfer them from an old KDCFILE using KDCUPD.

TRUSTED

Messages between the client and the application are not encrypted.
A "trusted" client can also start services whose service TACs request encryption (generated with TAC ENCRYPTION-LEVEL=2 | 5). This means that every client that connects via this LTERM pool is considered to be a trusted client.

TRUSTED should only be selected for an LTERM pool if communication is conducted through a secure connection.

FORMAT=

This operand is supported on BS2000 systems only.

Start format for users on terminals that sign on to the application via this LTERM pool (see also the statement LTERM ...,FORMAT=, in section "LTERM - define an LTERM partner for a client or printer").
Once the connection is established, the format specified in *formatname* is output on the terminal, provided a terminal-specific restart has not been performed.

Default: No start format

IDLETIME=

time

The maximum time in seconds that openUTM may wait for input from the client outside of a transaction, i.e. after the end of a transaction or after signing on. If this time is exceeded, then openUTM clears down the connection to the client. If the client is a terminal, then message K021 is output before the connection is cleared.

This function serves to improve data security:
If a user forgets to sign off from the terminal when taking a break or when finishing his or her work on the terminal, then the connection to the terminal or client is automatically cleared down after the wait time has run out. This reduces the chance of someone gaining unauthorized access to the system.

Default: 0 (= no wait time limit). For TPOOLs that HTTP clients can connect to, the default value is 180 seconds.
Maximum value: 32767
Minimum value: 60

If you specify a value that is greater than zero and smaller than the minimum value, KDCDEF replaces the value with the minimum value.

KERBEROS-DIALOG =

This operand is supported on BS2000 systems only.

Y

A Kerberos dialog is performed when a connection is established for terminals that support Kerberos and that connect to the application directly via this terminal pool (not via OMNIS).

openUTM stores the Kerberos information in the length resulting from the maximum lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated to this length and stored.
If a length greater than zero is generated neither for MAX PRINCIPAL-LTH nor for MAX CARDLTH, a warning message is issued.

The KDCS call INFO (KCOM=CD) allows a program unit run to read this information.
Exception: A user has subsequently signed on to this client with an ID card.
In this event, the Kerberos information is overwritten by the card ID information.

N

No Kerberos dialog is performed,
Default.

KSET= keysetname1

Name of a key set assigned to this LTERM pool. The key set must be defined with a KSET statement.

This defines the access permissions for the LTERM partners of this LTERM pool with respect to using the services of the application and remote services (LTACs) generated in this application.

An LTERM partner of this LTERM pool can only be used to start services of the application that are protected with a lock code or an access list and only address remote services that are protected with a lock code or an access list if the following applies: The key set assigned to the LTERM partner and the KSET of the UTM user ID under which sign-on using this LTERM partner was performed must contain the key code or access code that matches the lock code or access list.

With PTYPE=APPLI, SOCKET, UPIC-R, UPIC-L the following additionally applies with respect to the key set of the user ID:

- If the client does not pass a real user ID to openUTM for the session/conversation, then its access rights are the result of the set of key codes that are contained in the key set generated with KSET and the key set generated with USER-KSET. The key set *keysetname1* should therefore contain all key codes that are also contained in the key set generated with USER-KSET.
- If the client passes a user ID, then its access rights are the result of the set of key codes that are contained in the key set of the user ID and the key set generated with KSET.

LOCALE= (lang_id,terr_id,ccsname)

This operand is supported on BS2000 systems only.

Language environment of clients that sign on to the UTM application via the LTERM pool.

lang_id Freely selectable language identifier for the clients of the LTERM pool, up to two characters in length.
The language identifier may be queried by the program units of the application, so that messages can be sent to the terminals in the client’s language.

terr_id Freely selectable territorial identifier for the clients of the LTERM pool, up to two characters in length.

The territorial identifier may be queried by the program units of the application, so that any special territorial features of the client’s language can be taken into consideration in messages.

ccsname

(**c**oded **c**haracter **s**et **name**)
Name of an extended character set (CCS name) up to eight characters in length. The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the XHCS User Guide). The character set must be compatible with an ISO character set supported by all terminals in the LTERM pool.
During generation, KDCDEF cannot check the validity of the CCS name under the BS2000 system or the compatibility condition.

The character set with the specified name is used for:

- outputting dialog messages on 8-bit terminals if the application is generated without user IDs, or if a user has not yet signed on to the LTERM partner of the LTERM pool and another CCS name has not been explicitly selected using an edit profile or a format.
- outputting asynchronous messages on 8-bit terminals if another CCS name is not explicitly selected using an edit profile or a format.

Default: If TPOOL ...,LOCALE is not specified, then the locale of the application defined in the MAX statement is used.

LOCK=

lockcode

Access protection to the LTERM pool. Lock code assigned to the LTERM partners of the LTERM pool. *lockcode* is a numeric value between 1 and the maximum value permitted in the application (MAX ...,KEYVALUE=). You can only sign on to the application on an LTERM partner of this LTERM pool under a UTM user ID (USER) for which a key set was generated with a key code that matches the lock code of the LTERM pool.

Default: 0 (the LTERM pool is not secured with a lock code)
Maximum value: Value of KEYVALUE defined in the MAX statement

LTERM=

ltermprefix

Prefix for the names of LTERM partners of the LTERM pool. LTERM names are eight characters in length, and consist of the prefix specified here followed by a serial number between 1 and the value defined for NUMBER=*number1*.

The maximum length of *ltermprefix* depends on the number of decimal places in *number1*. The number of characters in *ltermprefix* plus the number of decimal places in *number1* must be less than 8.

When specifying *ltermprefix* and *number1*, please note that LTERM partner names must be unique within the application. This applies for names generated with TPOOL ...,LTERM= (in all TPOOL statements) and for names defined in LTERM statements.

Example
With *number1*=1000 and LTERM=LTRM, the LTERM partners defined for the LTERM pool are assigned the names LTRM0001,LTRM0002,...,LTRM1000.

These names must not be specified in any LTERM statement.

> The specified names must not be assigned to any other object in name class 1. See also section "Uniqueness of names and addresses".

MAP=

Controls the code conversion (EBCDIC <-> ASCII) for the user messages exchanged between the communication partners. User messages are passed in the message area on the KDCS interface in the message handling calls (MPUT/MGET/FPUT/DPUT/FGET).

USER

openUTM does not convert the data of the message area, i.e. the messages are transferred between the partners application unchanged.
Note that the user message contains the transaction code in the case of TS applications (partners with PTYPE=SOCKET or APPLI). It must be encoded in the form that the receiving system expects, i.e. on BS2000 systems in EBCDIC and in ASCII on Unix, Linux and Windows systems.

For TPOOLs, to which exclusively HTTP clients may connect, only the default value USER may be specified for parameter MAP. On BS2000 systems a code conversion for HTTP clients can be configured using the statements CHAR-SET and HTTP-DESCRIPTOR, see the description in chapters "CHAR-SET- assign names to code tables (BS2000 systems)" and "HTTP-DESCRIPTOR - define a HTTP Descriptor".

Default: USER

SYSTEM | SYS1 | SYS2 | SYS3 | SYS4

This parameter is only permitted for the following partners:

- BS2000 systems: partners with PTYPE=SOCKET
- Unix, Linux and Windows systems: partners with PTYPE=SOCKET or APPLI

UTM converts the user messages based on the conversion tables provided for the code conversion (see section "Code conversion"), i.e.:

- Prior to sending, the code is converted from ASCII to EBCDIC on Unix, Linux and Windows systems and from EBCDIC to ASCII on BS2000 systems.
- After receipt, the code is converted from EBCDIC to ASCII on Unix, Linux and Windows systems and from ASCII to EBCDIC on BS2000 systems.

The specifications SYSTEM and SYS1 are synonymous.

UTM assumes that the messages contain only printable characters.

NETPRIO=

This operand is supported on BS2000 systems only.

Transport priority to be used on the transport connections assigned to this LTERM pool.

NETPRIO is not relevant when the connection from the partner application is established via the socket interface (transport protocol SOCKET).

Default: MEDIUM

NUMBER=

number1

Maximum number of LTERM partners in this LTERM pool. Up to *number1* clients can then sign on to the application via the LTERM pool. The maximum value permitted for *number1* depends on the number of names generated in the UTM application (see section "Number of names").

Minimum value: 1

PRONAM= System on which the clients must be located in order to sign on to the application via this LTERM pool.

Unix, Linux and Windows systems:

- PRONAM= may only be specified for LTERM pools of type PTYPE=APPLI, SOCKET or UPIC-R.
- No distinction is made between uppercase and lowercase notation; KDCDEF always converts the name of the partner computer into uppercase.
- The combination of PRONAM/PTYPE/BCAMAPPL must be unique.
- Default value for PTYPE=TTY and UPIC-L: blanks

BS2000 systems:

- PRONAM= is a mandatory operand.
- If PROTOCOL=STATION is set, the combination of PRONAM/PTYPE/BCAMAPPL must be unique.
- If PROTOCOL=NO is set, the combination of PRONAM/BCAMAPPL must be unique.

{ processorname | C’processorname’ }

Name of the partner computer.

The complete name (FQDN) by which the computer is known in the DNS must be specified.

The name can be up to 64 characters long.

Only clients located on this system can sign on to the application via this LTERM pool.

If *processorname* contains special characters it must be entered in the form of a character string using C’...’ .

Instead of the name of the partner computer name you can enter the mapped name of a SUBNET statement. These names begin with a "*". All clients which belong to the subnet defined with this SUBNET statement can connect themselves to a TPOOL generated in this way. Please note that in this case you must specify the local application name of the associated SUBNET statement in the BCAMAPPL operand.

*ANY Any client that fulfills the following conditions can sign on to the application via the LTERM pool:

- The client must not be explicitly generated in a PTERM statement.
- The client type must match the entry in PTYPE.
- Another LTERM pool must not generated for the system on which the client is located or for the same client type. This prevents open LTERM pools from being used as an “overflow” for other LTERM pools.

PROTOCOL= This operand is supported on BS2000 systems only.

This operand specifies whether or not the user services protocol (NEABT) is to be used between the UTM application and the clients accessed via this LTERM pool.

N

openUTM does not use a user services protocol.
If PROTOCOL=N is generated, it is not possible to establish connections to terminals in this LTERM pool via a multiplex connection (see the description of the MUX statement in section "MUX - define a multiplex connection (BS2000 systems)").

PROTOCOL=N must be generated for UPIC client programs (PTYPE=UPIC-R) and for TS applications (PTYPE=APPLI or SOCKET). In this case, openUTM ignores the entry PROTOCOL=STATION without outputting a UTM message.

If you specify PTYPE=*ANY, openUTM ignores the entry PROTOCOL=NO.

STATION

The user services protocol (NEABT) is used between the UTM application and the clients accessed via this LTERM pool.

With PTYPE=*ANY, you must specify PROTOCOL=STATION. In this case, openUTM requires the user services protocol (NEABT) to determine the partner type if this is not explicitly specified during generation (PTYPE=*ANY).

Default:
N if PTYPE=APPLI, SOCKET or UPIC-R
STATION if PTYPE!=APPLI, SOCKET or UPIC-R.

PTYPE=

Type of client that can sign on to the application via this LTERM pool.

If you have specified an application name in BCAMAPPL= that is generated for communication via the socket interface (BCAMAPPL statement with T-PROT=SOCKET), then you must set PTYPE=SOCKET.

This is a mandatory operand.

partnertyp

Type of client.

A list of partner types supported can be found in the description of the PTERM control statement in section "PTERM - define the properties of a client/printer and assign an LTERM partner". Please note that printers **cannot** be connected via LTERM pools.

*ANY

only permitted on BS2000 systems.
PTYPE=*ANY describes an open LTERM pool. All clients that support the user services protocol (PROTOCOL=STATION) and that are located on the processor defined with PRONAM= may connect to this LTERM pool.

In this case, openUTM takes the partner type from the user services protocol during connection setup. Only then can it be determined whether or not the partner type is supported.

The advantage of PTYPE=*ANY is that it allows you to include clients in the configuration without having to know their type. The configuration is also easier to maintain because even if the type is modified in the terminal emulation, for example, this client can still sign on to the application without having to modify the KDCDEF generation.

QLEV= queue_level_number

(**q**ueue **lev**el)
Maximum number of asynchronous messages that can be accommodated in the message queue of the LTERM partner. If this threshold value is exceeded, openUTM rejects all further FPUT calls for this LTERM partner with UTM message 40Z.

Default: 32767
Minimum value: 0
Maximum value: 32767

If you exceed the maximum value, KDCDEF automatically resets your entry to the default value without outputting a UTM message.

STATUS= NUMBER=*number1* defines the number of LTERM partners in the LTERM pool. STATUS=*number2* defines the number of clients that are unlocked (ON) and locked (OFF) for the LTERM pool when the application is started.
The status can be modified by administration during operation.

Default:
STATUS=(OFF , 0), i.e. all clients of the LTERM pools are unlocked.

ON *number2* clients are unlocked.

OFF *number2* clients are locked.

number2 Number of clients (and thus the number of LTERM partners of the LTERM pool) which are locked or unlocked.

TERMN= termn_id

Identifier up to two characters in length, which indicates the type of client. openUTM provides this identifier to the application program in the KCTERMN field of the KB header.

termn_id is not queried by openUTM, but can be used by the user for analysis purposes.

Default values:

If this operand is not specified, openUTM sets the KCTERMN field to the default ID of the partner type specified in the PTYPE operand. However, the user can select other values if desired.

The default values are listed in the partner type table for the PTYPE operand of the PTERM statement "PTERM - define the properties of a client/printer and assign an LTERM partner".

BS2000 systems:
If TERMN is not explicitly specified for clients generated with PTYPE=*ANY, openUTM does not enter the terminal mnemonic in KCTERMN until the connection is established. This is the default terminal mnemonic of the type specified in the user services protocol of the connection request.

USER-KSET= keysetname2

This is only allowed if the application is generated with user IDs and PTYPE=APPLI, SOCKET, UPIC-R or UPIC-L is specified. You may only set USER-KSET= in conjunction with KSET= .

You must specify the name of a key set for *ksetname2*. The key set must be defined with a KSET statement.

You specify the minimum access rights that a client connected via this LTERM pool can exercise with USER-KSET= .

ksetname2 takes effect when the client is signed on under the connection user ID. Its access rights are the result of the set of key codes that are contained in the key set generated with KSET= and in the key set generated with USER-KSET= (intersection). For this reason, all key codes contained in USER-KSET=*ksetname2* should also be contained in KSET=*ksetname1*.

Default: No key set
The access rights specified in KSET are always valid.

USP-HDR= Specifies the output messages for which openUTM is to create a UTM socket protocol header for the connections generated with this statement.

A value that is not equal to NO may only be specified with LTERM pools for which communication is configured via socket connections (PTYPE=SOCKET).

A description of the USP header can be found in the openUTM manual „Programming Applications with KDCS”.

For TPOOLs to which exclusively HTTP clients may connect, only the default value NO may be specified for parameter USP-HDR.

ALL For all output messages (dialog, asynchronous, K messages) openUTM creates a UTM socket protocol header and adds this to the front of the message.

MSG openUTM only creates a UTM socket protocol header and adds this to the front of the message for K messages only.

NO No UTM socket protocol headers are created.

Default: NO

6.5.55 TRANSFER-SYNTAX - define the transfer syntax

You only need the TRANSFER-SYNTAX control statement when you want to define your own application context for communication based on the OSI TP protocol (see the APPLICATION-CONTEXT statement in section "APPLICATION-CONTEXT - define the application context").

It allows you to define a local name for a transfer syntax, and to assign an object identifier. The transfer syntax determines the rules governing the encoding and decoding of the abstract syntax defined in the object identifier.

The transfer syntax defined here must be supported by the OSS version used. OSS only supports the BER transfer syntax at the present time.

TRANSFER-SYNTAX	`transfer_syntax_name` `,OBJECT-ID`ENTIFIER=`object_identifier`

transfer_syntax_name

Local name for a transfer syntax up to eight characters in length. This name must be unique within the UTM application.

The *transfer_syntax_name* BER (Basic Encoding Rules) is reserved.

OBJECT-IDENTIFIER=

object_identifier

Object identifier of the transfer syntax specified as follows:

object_identifier=(number1,number2, ... ,number10)

number is a positive integer in the range 0 to 67108863. For *object_identifier*, you can specify two to ten integers enclosed in parentheses, each of which is separated by a comma. The number of integers entered and their positions are relevant.
Instead of the integer itself, you can also specify the symbolic name assigned to this integer. The table in section "OSI terms" shows the permitted values for *number* at the various positions.

object_identifier must be unique within the UTM application, i.e. another transfer syntax must not be generated with the same object identifier.

openUTM generates the BER transfer syntax by default:

```
TRANSFER-SYNTAX BER,                          -
                OBJECT-IDENTIFIER=(2, 1, 1)   -
```

Symbolic description of the object identifier:

```
(joint-iso-ccitt, ansl, basic-encoding)
```

6.5.56 ULS - define a name for a ULS block

Each UTM user ID can be assigned a user-specific long-term storage area (ULS), which can contain several blocks each of which is addressed by means of a name.

The ULS control statement allows you to define a name for a ULS block. openUTM then provides each UTM user ID with a ULS block with this name. By issuing several ULS statements with different block names, you can define several blocks.

In the case of distributed processing based on LU6.1, the ULS blocks defined in a ULS statement are also assigned to sessions (LSES).

This statement is required only if the application is generated with user IDs.

You can issue up to 100 ULS statements.

Note for UTM cluster applications on Unix, Linux and Windows systems:
If you modify, remove or add ULS statements then you must regenerate both the initial KDCFILE and the UTM cluster files by specifying GEN=(CLUSTER,KDCFILE) in the OPTION statement.

`ULS`	`name`

name Name of a ULS block up to eight characters in length, which can be used to address the block from a program unit.

6.5.57 USER - define a user ID

The USER control statement allows you to define user IDs for the UTM application. These are then used by users and client programs to sign on to the application. The following can be defined for user IDs:

- the authentication procedure (password, magnetic strip card on BS2000 systems and Unix)
- complexity level and period of validity of password
- access rights (lock/key code or access list concept)
- administration authorization
- the user status
- properties of the USER queue that belongs to the user ID
- the start format
- UTM SAT administration authorization (BS2000 systems)
- the user-specific language environment (BS2000 systems).
- Method and type of authentication (BS2000 systems: password, magnetic strip card, Kerberos principal)

At least one user ID must be assigned administration authorization in order to manage the application. Administration authorization can be granted to several user IDs, thereby enabling several users to simultaneously call administration functions under the respective user ID. On BS2000 systems the same is true for the UTM SAT administration authorization and calling SAT preselection functions.

An application can also be generated without user IDs. In this case, users are not required to identify themselves, and openUTM uses the name of the respective client internally as the user ID. All users can thus issue administration commands and on BS2000 systems UTM SAT administration commands. If you work without user IDs, openUTM will not be able to use some data protection functions.

USER	username [,KSET=keysetname] [,PASS={ (password,**D**ARK) | (*RANDOM,**D**ARK) | password | *RANDOM }] [,KSET=keysetname] [,PERMIT={ NONE |ADMIN | SATADM[1] | (ADMIN,SATADM)[1] }] [,PROTECT-PW=([length] ,[{ NONE | MIN | MED | MAX }] ,[maxtime] ,[mintime])][2] [,QLEV5=queue_level_number] [,QMODE = { STD | WRAP-AROUND }] [,Q-READ-ACL=read-keysetname] [,Q-WRITE-ACL=write-keysetname] [,RESTART={ **Y**ES | **N**O }] [,STATUS={ ON | OFF }] ***additional operands on BS2000 systems*** [,CARD=(position,characterstring)] [,FORMAT= { + | * | # }formatname] [,LOCALE=([lang_id][,[terr_id][,ccsname]])] [,PRINCIPAL=characterstring] [,SATSEL={ NONE | BOTH | SUCC | FAIL }]

[1] only on BS2000 systems

[2] Commas at the end can be omitted, i.e. you can specify (8,NONE) instead of (8,NONE,,).

username

UTM user ID specified by the user when signing on to the application, or by a client when opening a conversation with the application. *username* can be up to eight characters in length.
(See also the operand USER=*username* in the LTERM statement starting in section "LTERM - define an LTERM partner for a client or printer").

The specified name must be unique and must not be assigned to any other object in name class 2. See also section "Uniqueness of names and addresses".

If *username* is identical to the name of an LTERM that is assigned to a PTERM with PTYPE=APPLI, SOCKET or UPIC-R, you must bear in mind the notes detailed under LTERM.

CARD=

This operand is supported on BS2000 systems only.
specifies whether a magnetic strip card is to be checked when signing on to the application under this user ID, and defines the ID card information to be verified. Enter a subfield of the information stored on the magnetic strip card, which is to be checked by openUTM.

The following must apply for this parameter:
pos + length(string) -1 <= MAX CARDLTH
Otherwise the parameter is ignored.

You cannot specify PRINCIPAL= if you have specified CARD=.

position

Start position of the ID card information to be checked:
position = 1 corresponds to the first byte, etc.

characterstring

When signing on to the application, openUTM checks whether the ID card information starting at the defined position begins with this character string.

characterstring can be specified in the following format:

- as a hexadecimal character string; hexadecimal characters always occur in pairs, e.g. X'DDEF'
- as an alphanumeric character string, e.g. FRIDOLIN or C'@FRIEDEL'.

Special characters must be entered in the format C'...' or X'...'.

Default: No ID card check performed when signing on to the application Maximum length: 100 bytes (see also MAX ...,CARDLTH=)

FORMAT=

This operand is supported on BS2000 systems only.

Format identifier for a user-specific start format.
This start format is automatically output after each successful attempt to sign on to the application, provided an open service does not exist for this user. However, if an open service exists for the user (USER) following a successful sign-on check, the start format is **not** displayed, rather the last dialog screen appears (service restart). If you use your own sign-on procedure, the name of the user-specific start format may be queried in the second part of the sign-on procedure using the SIGN ST call.

The format identifier composes as follows:

+, * or # followed by an alphanumeric name (*formatname*) up to seven characters in length.

#formats can only be used in the context of a sign-on procedure.

The terms have the following meanings:

+

When the next MGET call of the program unit is issued, each entry in a format field is preceded by 2 bytes for the attribute field in the KDCS message area, i.e. the field properties can be modified by the program unit.
The format identifier at the KDCS interface is thus +*formatname*.

*	When the next MGET call of the program unit is issued, the entry in a format field is not preceded by any bytes for an attribute field in the KDCS message area, i.e. the field properties cannot be modified by the program unit. The format identifier at the KDCS interface is thus **formatname*.
#	This identifies a format with extended user attributes. The field properties and global format properties can be modified by the program unit. The format identifier at the KDCS interface is thus #*formatname*. Default: No start format
KSET=	keysetname Name of the key set assigned to the user ID. The key set is defined with the KSET statement. A maximum of one key set can be assigned per USER. The key set defines the access permissions for this user ID with respect to using the services of the application and remote services (LTACs) generated in this application. This user ID can only be used to start services of the application that are protected with a lock code or an access list and only address remote services that are protected with a lock code or an access list if the following applies: The key set assigned to the user ID and the key set of the LTERM partner under which sign-on using this user ID was performed must contain the key code or access code that matches the lock code or access list. The lock/key code concept and the access list concept are both described in detail in the openUTM manual “Concepts und Functions”. An introduction to data access control can be found as of "Data access control". Services whose service TACs are not secured with codes can be called by the user or the client program without restriction. Further information on the lock/key code concept can be found in the openUTM manual “Concepts und Functions”. Default: No key set, i.e. the user can only access clients and LTERM partners that have not been secured with lock codes.
LOCALE=	(lang_id,terr_id,ccsname) This operand is supported on BS2000 systems only. Language environment of the user
lang_id	Freely selectable language identifier for the UTM user ID, up to two characters in length. The language identifier may be queried by the program units of the application, so that messages can be sent by the program units to the user ID in the user’s language.

terr_id

Freely selectable territorial identifier for the UTM user ID, up to two characters in length.

The territorial identifier may be queried by the program units of the application, so that any special territorial features of the user’s language can be taken into consideration in messages to the user.

ccsname

(**c**oded **c**haracter **s**et **name**)
Name of an extended character set (CCS name) up to eight characters in length.
The specified CCS name must belong to one of the EBCDIC character sets defined under the BS2000 system (see also the "XHCS User Guide").
During generation, openUTM cannot check the validity of the CCS name under the BS2000 system.

The character set with the specified CCS name is used for outputting dialog messages, provided the user is signed on to an 8-bit terminal and another CCS name is not explicitly selected using an edit profile or a format.
The character set must be compatible with an extended ISO character set supported by the terminal. During generation, openUTM cannot check this compatibility condition, i.e. incorrect entries cannot be intercepted by KDCDEF.

Default:
Locale of the application defined in the MAX statement is used if USER ...,LOCALE is not specified.

PASS=

Password up to 16 characters in length which must be specified by the user during the sign-on check. This password must comply with the level of complexity defined in the PROTECT-PW= operand.

PASS= may not be specified together with PRINCIPAL=.

If you enter *RANDOM here, a secret random password is generated for the user ID. A valid password must then be transferred to the user ID using the KDCUPD tool or by means of administration. Passwords created in this way are not subject to the conditions set in PROTECT-PW=.

> **i** Make sure that at least one user ID is configured with administration authorization by the startup time at the latest. This user ID must not be assigned a password created using *RANDOM, as otherwise the application cannot be administered.

BS2000 systems:
password can be entered in the following format:

- hexadecimal, e.g. X'DDFF'
- as a constant, e.g. C'@LKE'
- as a printable alphanumeric character string, e.g. NORBERT.

Unix, Linux and Windows systems:
The password can be entered in the form of a printable alphanumeric character string, e.g. `UTM4EVER` or `C'UTM_ever'`.

Default: 16 blanks (i.e. no password)

The parameters have the following effects on the signon procedure:

password
*RANDOM

BS2000 systems:
Standard sign-on dialog:
The user must enter 'KDCSIGN *username*,*password*' to sign on to the application.

Sign-on procedure:
The user ID and password must be transferred to openUTM using the SIGN ON call.

Unix, Linux and Windows systems:
Standard sign-on dialog:
To sign on to the application, the user first enters *username*. openUTM then prompts the user to enter the password.

Sign-on procedure:
In an intermediate dialog, openUTM prompts the user to enter the password as in the standard sign-on procedure.

(password, DARK) (*RANDOM, DARK)	*BS2000 systems*: Standard sign-on dialog: To sign on to the application the user first enters ’KDCSIGN *username*’. openUTM then prompts the user to enter the password in a blanked-out field on the screen. Sign-on procedure: In an intermediate dialog, openUTM prompts the user to enter the password in a blanked-out field. *Unix, Linux and Windows systems*: Standard sign-on dialog: To sign on to the application, the user first enters *username*. openUTM then prompts the user to enter the password. Sign-on procedure: In an intermediate dialog, openUTM prompts the user to enter the password as in the standard sign-on procedure.
PERMIT=	Administration authorization level of the user within the local application
ADMIN	The user can execute administration functions under this user ID.
NONE	The user must not execute any administration functions. Default: NONE On BS2000 systems the user also must not execute any SAT preselection functions.
SATADM	The user can execute SAT preselection functions (UTM SAT administration).
(ADMIN,SATADM)	The user can execute administration and SAT preselection functions.

PRINCIPAL= characterstring

This operand is supported on BS2000 systems only.

Authentication of the user is to be performed using Kerberos. It is only possible to authenticate users using Kerberos if the user signs in directly (not via OMNIS) at a terminal that supports Kerberos.

openUTM stores the Kerberos information in the maximum of the lengths generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated and stored in this length.

The KDCS call INFO (KCOM=CD) enables a program unit to read this information as long as the user is signed in on this client.

Specifying PRINCIPAL excludes the possibility of specifying the parameters CARD and PASS.

characterstring must be specified as follows as an alphanumeric string enclosed in single quotes:

C'windowsaccount@NT-DNS-REALM-NAME'

windowsaccount:

Domain account of the user

NT-DNS-REALM-NAME:

DNS name of the Active Directory domain. This name is a fixed value for every Active Directory domain and was assigned when the Kerberos key was set up.

The length of the character string passed must not be greater than the value specified for MAX PRINCIPAL-LTH. Otherwise the parameter is ignored.

openUTM stores the Kerberos information in the length resulting from the maximum length generated for MAX PRINCIPAL-LTH and MAX CARDLTH. If the Kerberos information is longer, it is truncated to this length and stored.

The KDCS call INFO (KCOM=CD) allows a program unit run to read this information as long as the user is signed in on this client.

Maximum length: The value generated with MAX ...,PRINCIPAL-LTH. See "MAX - define UTM application parameters".

Default: No Kerberos authentication

PROTECT-PW= Specifies the minimum length, level of complexity, and minimum and maximum validity period of the user password. The values defined for PROTECT-PW must be taken into consideration when specifying the password in the PASS= operand. They are checked by openUTM when the password is changed by the administrator (KDCUSER administration command, see the openUTM manual “Administering Applications”) or by a program unit (SIGN CP call).

length

Minimum number of characters that must be contained in the password.
The administrator can only delete the user password if the value 0 is specified for *length*.

Default: 0

Minimum value:
0 for NONE or if a level of complexity is not defined
1 for MIN
2 for MED
3 for MAX

Maximum value: 16

NONE/MIN/MED/MAX

Level of complexity of the password

NONE

The password can be any character string.

Default: NONE

MIN

In the password, up to two consecutive characters may be identical. The minimum length of the password is one character.

MED

In the password, up to two consecutive characters may be identical. The password must contain at least one letter and one number. The minimum length of the password is two characters.

MAX

In the password, up to two consecutive characters may be identical. The password must contain at least one letter, one number, and one special character. The minimum length of the password is three characters. Special characters are all characters other than a-z, A-Z, 0-9, and blanks.

maxtime

Maximum validity period:
maxtime specifies the maximum number of days for which the password is valid.

If a validity period is specified, then the validity of the password expires at the end of the last day of the specified validity period. For instance, if the validity period is one day, the password ceases to be valid at 24:00 hours on the following day.

If the application is generated with SIGNON GRACE=YES, when the application is regenerated the password is set to “expired”, the user must then assign a new password the first time they sign on.

If the password expires, then the next action taken depends on how the UTM application is generated:

- With grace sign-ons (SIGNON GRACE= YES)
  The user can and must change the password the next time the they sign on, as long as the sign-on service of the application offers them this opportunity. If this is not the case, the password must be modified by administration otherwise the user will no longer be able to sign on under this user ID. This may occur, for example, with users that sign on via TS applications and UPIC clients without sign on services or via an OSI-TP partner.
- Without grace sign-ons (SIGNON GRACE= NO)
  openUTM rejects a sign-on attempt with message K120. The administrator must then change the password.

With *maxtime* = 0 the validity period of the password is not restricted.

Default: 0 (validity period not restricted)
Maximum value: 180
Minimum value: 0

mintime

Minimum validity period:
You specify the minimum validity period of the password in days in *mintime*.
Once the user has changed the password, the user may only change the password again after the minimum validity period has expired.

By specifying *mintime* > 0 you can prevent a user whose password has expired from changing his or her password twice in a row to set the password back to the original (= expired) password.

If a minimum validity period of one day is specified, then the password may be changed no earlier that at 12.00 midnight of the following day (local time of the generation).

The user can always change the password after the administrator has changed the password and after a new generation, regardless of whether the minimum validity period has expired or not.

mintime must not be larger than *maxtime* (maximum validity period).
If *mintime*=0 is specified, then the minimum validity period of the password is not restricted.

Default: 0 (no limit)
Minimum value: 0
Maximum value: 180

QLEV=

queue_level_number

(**q**ueue **lev**el)
Specifies the maximum number of asynchronous messages that may be buffered in the message queue of the user (= USER queue). QLEV can be used to make sure that the page pool is not overloaded with messages for this USER.
openUTM only takes asynchronous jobs into account at the end of the transaction. It is thus possible that the maximum number of messages for a message queue as specified in QLEV may be exceeded if several messages are created for this queue during a single transaction.

If the threshold value has been exceeded, then the behavior will depend on the value set in the operand QMODE=, see below.

With QLEV=0 no messages may be saved in the queue and with QLEV=32767 the queue length is not restricted.

Default: 32767
Minimum value: 0
Maximum value: 32767

If a value is specified that is greater than the maximum, this is set back to the default in the KDCDEF run. No message is issued.

QMODE =

(**Q**ueue **Mode**)
Determines the behavior of openUTM in the event that the maximum permitted number of messages that may be saved in the USER queue has been reached and thus the queue level (QLEV= operand) has also been reached.

STD — When the queue level is reached openUTM rejects all additional messages for the queue with negative return code (40Z for DPUT).

WRAP-AROUND — openUTM continues to accept messages for the queue, even if the queue level has been reached. When a new message is written to the queue, openUTM deletes the oldest message in the queue and replaces it with the new one.

Default: STD

Q-READ-ACL= read-keysetname

Specifies the read and delete rights for external users in the USER queue. In *read-keysetname* you must enter a key set that has been generated using a KSET statement.

If you enter Q-READ-ACL=, an external user (!= *username*) is only permitted read access to the queue if both the key set of their user ID and the key set of the LTERM partner via which the user is signed on contain at least one of the key codes contained in the key set *read-keysetname*.

The owner (*username*) of the USER queue always has read and delete rights to their queue, even if the rights are restricted using Q-READ-ACL.

If you do not specify Q-READ-ACL=, all users have both read and delete rights in the queue.

Default: no key set

Q-WRITE-ACL= write-keysetname

Specifies the write rights for external users in the USER queue.
In *write-keysetname* you must enter a key set that has been generated using a KSET statement.

If you enter Q-WRITE-ACL=, an external user (!= *username*) is only permitted write access to the queue if both the key set of their user ID and the key set of the LTERM partner via which the user is signed on contain at least one of the key codes contained in the key set *write-keysetname*.
The owner (*username*) of the USER queue always has the write rights to their queue, even if the rights are restricted using Q-WRITE-ACL.

If you do not specify Q-WRITE-ACL= all users have both write rights in the queue.

Default: no key set

RESTART= This specifies whether openUTM is to save the service data for a user ID so that a service restart will be possible on the next sign-on under this user ID.

YES

The service context belonging to this user ID is saved. This means that a service restart can be performed for users who sign on using this user ID if an open service exists for the user ID.

With a service restart, the type of the client and possibly the generated sign on service may play a role. Additional information can be found in section "Generating a restart" and in the openUTM manual “Using UTM Applications”.

Default: YES

NO

The service context belonging to this user ID is not saved, no service restart is possible,

- If the connection is shut down during operation by KDCOFF, if it is lost, or if the application is terminated normally, the service is rolled back to the last synchronization point and terminated. The event exit VORGANG is then called with KCKNZVG=D (=Disconnect).
- During a UTM warm start following abnormal termination of the application, an open service for this UTM user is terminated without calling the event exit VORGANG.
- Following connection setup, KDCDISP/KDCLAST behaves in the same way as after regeneration.

If RESTART=NO is specified together with SIGNON MULTI-SIGNON=YES, several users can sign on simultaneously to openUTM under this user ID, but only one user can sign on to the terminal. Conversely, it is possible for any number of client programs can sign on simultaneously.

> ¡ Explicitly generated connection user IDs to UPIC clients are always generated with RESTART=NO (without any message) .

SATSEL=

This operand is supported on BS2000 systems only.

SAT logging mode for this user

If SAT logging is activated (MAX SAT=YES), all events triggered by this user are logged as defined in this operand.

The SATSEL control statement is used to define the general SAT logging mode for all TACs and users. This can be supplemented by the SATSEL operand of the USER statement, which allows you to define user-specific logging. If the logging of an event class is prohibited in the SATSEL statement, events of this class are not logged. (For information on the link between the EVENT-, TAC- and USER-specific log settings, see the openUTM manual “Using UTM Applications on BS2000 Systems”.)

SATSEL can be generated even if SAT logging is deactivated (MAX statement with SECLEV=NO and SAT=OFF). In this case, the statements are not effective when the application is started, but SAT logging is predefined. When required, SAT logging can then be activated during operation (UTM SAT administration command KDCMSAT, see the openUTM manual “Using UTM Applications on BS2000 Systems”).

NONE

A user-specific SAT logging mode is not defined.

Default: NONE

BOTH

Both successful **and** unsuccessful events are logged.

SUCC

Only successful events are logged.

FAIL

Only unsuccessful events are logged.

STATUS=

Status (locked or unlocked) of the user ID when the application is started.

ON

The user ID is unlocked.

Default: ON

OFF

The user ID is locked. It cannot be used by a user or client to sign on to the application until it has been released by the administrator.

> i User IDs that are implicitly or explicitly assigned to an UPIC client or a client of a TS application via an LTERM statement (LTERM ...,USER=) are always locked. They cannot be authorized by the UTM administrator. These user IDs are called connection user IDs.

6.5.58 UTMD - application parameters for distributed processing

The UTMD control statement allows you to define values for distributed processing throughout the application. A UTMD statement is only required for applications that use either the LU6.1 or the OSI TP protocol for communication.

The UTMD statement may only by entered once.

If you use the OSI TP protocol in your application, you can specify the Application Process Title (APT) of the application in the UTMD statement. This is required by some heterogeneous partners that support another variant of the OSI TP protocol in order to establish a connection. These applications expect the specification of the Application Process Title on establishment of the connection.

The application process title is combined with any application entity qualifier (AEQ) assigned to an access point of your application to form an application entity title (AET), which is unique throughout the OSI network. This is used by the partner application to identify the access point of the local application via which communication is to take place.

```
UTMD   [ APPLICATION-PROCESS-TITLE=object_identifier ]
       [ ,CONCTIME={ time1 | ( time1,time2 ) } ]
       [ ,MAXJR=%_maxjr ]
       [ ,PTCTIME=time3 ]
       [ ,RSET={ GLOBAL | LOCAL } ]
```

APPLICATION-PROCESS-TITLE=object_identifier

(only relevant if the OSI TP protocol is used in the application)
Address component of the application entity title (AET). The AET is required if you are working with transaction management (commit functional unit), or if a heterogeneous partner requires an AET to establish a connection.

object_identifier is the application process title (APT) of your application. Even if this is not defined by a standardization body, the relevant conventions for *component1* and *component2* must be observed when assigning an APT. Further information can be found in section "Application entity title (AET)" (OSI terms). In practice, the specified *object_identifier* must be unique within the network.

> If the application context (definition of the communication partner in the OSI-LPAP statement, "OSI-LPAP - define an OSI-LPAP partner for distributed processing based on OSI TP") agreed with a partner application contains the CCR syntax, you must enter an application process title here.

An application process title consists of at least 2 and at most 10 components. It is specified in the following format:

(*component1,component2,...,component10*)

The components are specified in the form of positive integers. Symbolic names are assigned to some numbers of individual components, and can be used instead of the numbers. In the application process title, both the number of components and their positions within the parentheses are relevant, e.g. (1,2,3), (1,2,3,0,0) and (0,1,2,3,0) identify different application process titles.

openUTM and the OSI standard only permit the following values or symbolic names for *component1*:
0 or CCITT
1 or ISO
2 or JOINT-ISO-CCITT

The values permitted for *component2* depend on the value of *component1*.

- If *component1* = 0 or 1, values between 0 and 39 are permitted for *component2* (0 <= *component2* <= 39).
- If *component1* = 2, values between 0 and 67108863 (2^{26}-1) are permitted for *component2* (0 <= *component2* <= 67108863).

Values between 0 and 67108863 (2^{26}-1) are permitted for all other components.

openUTM does not check whether the specified application process title is registered with a standardization body.

> *Note for UTM cluster applications on Unix, Linux and Windows systems*
>
> In the case of UTM cluster applications, the APT of the individual node applications is modified at node-specific level in order to ensure that the AET is unique. If the APT consists of fewer than 10 elements then the APT is extended by the index of the associated node when the node application is started. The index of a node is determined by the sequence of CLUSTER-NODE statements during generation.
>
> *Example:*
> If (1,2,3) is generated as the APT and if the UTM cluster application has two node applications, then the APT at runtime is as follows:
>
> (1,2,3,1) for node 1 (= first CLUSTER-NODE statement) and
> (1,2,3,2) for node 2 (= second CLUSTER-NODE statement) and
> If the generated APT already contains 10 elements then the APT remains unchanged for all node applications. In this case, links with OSI-TP implementations from other vendors may result in problems because the AET is not unique.

CONCTIME= (**con**nection **c**ontrol **time**)

time1 Maximum number of seconds for which openUTM monitors the opening of a session (LU6.1) or association (OSI TP). If the session or association is not opened within the specified time, openUTM shuts down the transport connection. This prevents the transport connection from being blocked if an attempt to open a session or association fails. This can occur if a message required to open the session/association is lost.

CONCTIME=0 for LU6.1 means opening is not monitored.
CONCTIME=0 for OSI TP means monitoring is set internally to 60 seconds.

Default: 0
Minimum value: 0
Maximum value: 32767

time2

Maximum number of seconds for which openUTM is to wait for confirmation from the partner application when sending an asynchronous message. Once the specified time has elapsed, openUTM shuts down the transport connection. The job is not lost however. Monitoring prevents the connection from being blocked because a confirmation has been lost, or because the loss of connection was not reported to openUTM by the transport system.
The value 0 means that monitoring is not performed.

Default: 0
Minimum value: 0
Maximum value: 32767

MAXJR=

%_maxjr

(**max**imal number of **j**ob **r**eceivers)
Specifies the maximum number of job-receiving services that can be addressed in the local application at any one time.
This corresponds to the number of APRO calls that can be active simultaneously.

The percentage value refers to the number of generated sessions and associations (maximum number of LSES statements for the LU6.1 protocol + total number of parallel connections specified in OSI-LPAP statements; ASSOCIATIONS operand). It must be within the range 0 to 200. If you enter a value > 100, APRO calls issued before the session is reserved can be entered in a table.

Default: 100,

i.e. the maximum number of job-receiving services active at a particular time is equal to the number of sessions and associations.

Minimum value: 0
Maximum value: 200

PTCTIME= time3

(**p**repare **t**o **c**ommit)
This is significant only for distributed processing via LU6.1 connections. PTCTIME defines the maximum number of seconds for which a job receiving service waits in PTC state (transaction status P) for confirmation from the job submitter. Once this time has elapsed, the connection to the job submitter is shut down, the transaction in the job-receiving service is rolled back, and the service is terminated. This can lead to inconsistent data if the transaction is committed in the partner application (mismatch). The value 0 means that the job-receiving service waits indefinitely for confirmation.

> If a value > 0 is specified in *time3* then this value is ignored by openUTM if a KDCSHUT WARN or GRACE has been issued. In this case, openUTM chooses the wait time in such a way that the transaction is rolled back before the application is terminated in order, if possible, to prevent the application from being terminated abnormally with ENDPET.

Default:
Value specified in MAX ...,TERMWAIT=*time* for the waiting time after PEND KP

Minimum value: 0
Maximum value: 32767

RSET= In the case of Distributed Transaction Processing, this operand defines how rolling back a local transaction affects the distributed transaction.

A local transaction can be rolled back:

- by a RSET call issued in a program unit, or
- by rolling back a database transaction involved in the local transaction.

GLOBAL After the local transaction is rolled back, the program unit must be terminated such that openUTM rolls back the distributed transaction.

Default: GLOBAL

LOCAL Rolling back the local transaction has no effect on the distributed transaction.

Inconsistencies may occur in the distributed databases if some of the local transactions involved in a distributed transaction are rolled back while others are concluded. If RSET=LOCAL is specified, it is no longer possible to guarantee global data integrity in the relevant system components. This is now the responsibility of the application program units. You must decide when it makes sense to terminate the distributed transaction and when to roll back the transaction.

6.6 Dialog control - effects of generation parameters

The following statements and parameters of the KDCDEF generation tool can be used to control the dialog during generation:

KDCDEF statement	Effect
EXIT ...,USAGE=INPUT	Event exit INPUT
LTAC ..., WAITTIME=	Maximum waiting time for distributed processing
LTERM ..., RESTART=	Refers to asynchronous messages and the service restart procedure if user IDs are not generated
LTERM ..., USER=	Automatic KDCSIGN
MAX ..., APPLIMODE=	Refers to the service restart procedure and asynchronous messages
MAX ..., CONN-USERS=	Application load: number of simultaneously active users or clients
MAX ..., NRCONV=	Maximum number of stacked services
MAX ..., PGWTIME=	Maximum time in seconds that a program unit is allowed to wait to receive messages after a blocked call.
MAX ..., TERMWAIT=	Maximum waiting time until the next input from the terminal in a multi-step transaction (following PEND KP)
PTERM ..., IDLETIME= TPOOL ..., IDLETIME=	Maximum waiting time until the next input from the client after the end of the transaction or after signing on (following PEND RE/FI/ER)
SIGNON ..., GRACE=	The user may or may not change his or her password after it has expired
SIGNON ..., MULTI-SIGNON=	Several users/clients may or may not be signed on at the same time under the same user ID
SIGNON ..., SILENT-ALARM=	Limits the number of unsuccessful sign-on attempts
USER statements defined	Sign-on check performed for all users
USER ..., PASS=	Sign-on check with a password; input may be blanked-out
USER ..., PROTECT-PW=	Authorization check with a blanked-out password, validity period, and level of complexity
USER ..., RESTART=	Service restart procedure

KDCDEF statement	Effect
SFUNC ...	Assignment of a F key or K key (on BS2000 systems) as a TAC, UTM command, or stacking request
TAC KDCBADTC	Event service BADTACS
TAC KDCSGNTC	Sign on using a sign-on procedure
On BS2000 systems the following parameters also are relevant	
LTERM ..., ANNOAMSG=	Asynchronous messages with or without prior announcement
LTERM ..., FORMAT=	LTERM specific start format
TPOOL ..., ANNOAMSG=	Asynchronous messages announced in advance before being output in the system line on the terminal
PTERM ..., CONNECT=	Automatic or non-automatic connection setup to the terminal by openUTM when starting the application
MAX ..., LOCALE=	Default language environment of the application
LTERM ..., LOCALE=	Language environment of clients that sign on via LTERM partners
TPOOL ..., FORMAT=	defines the start format for the terminal user
TPOOL ..., LOCALE=	Language environment of clients that sign on via LTERM pools
USER ..., CARD=	Sign-on check with a magnetic strip card
USER ..., FORMAT=	User specific start format
USER ..., LOCALE=	Language environment of the user
USER ..., PRINCIPAL=	User authentication will be performed using Kerberos.

6.7 Example generation: ComfoTRAVEL

The example generation *Comfo*TRAVEL is a travel reservation system which allows the customers of travel agents in Munich, Paris and New York, to book “all-inclusive” packages consisting of hotel, flight and leisure activities plus the necessary bank operations. To do this, the travel agents access a central reservation system (RMS Reservation Management System).

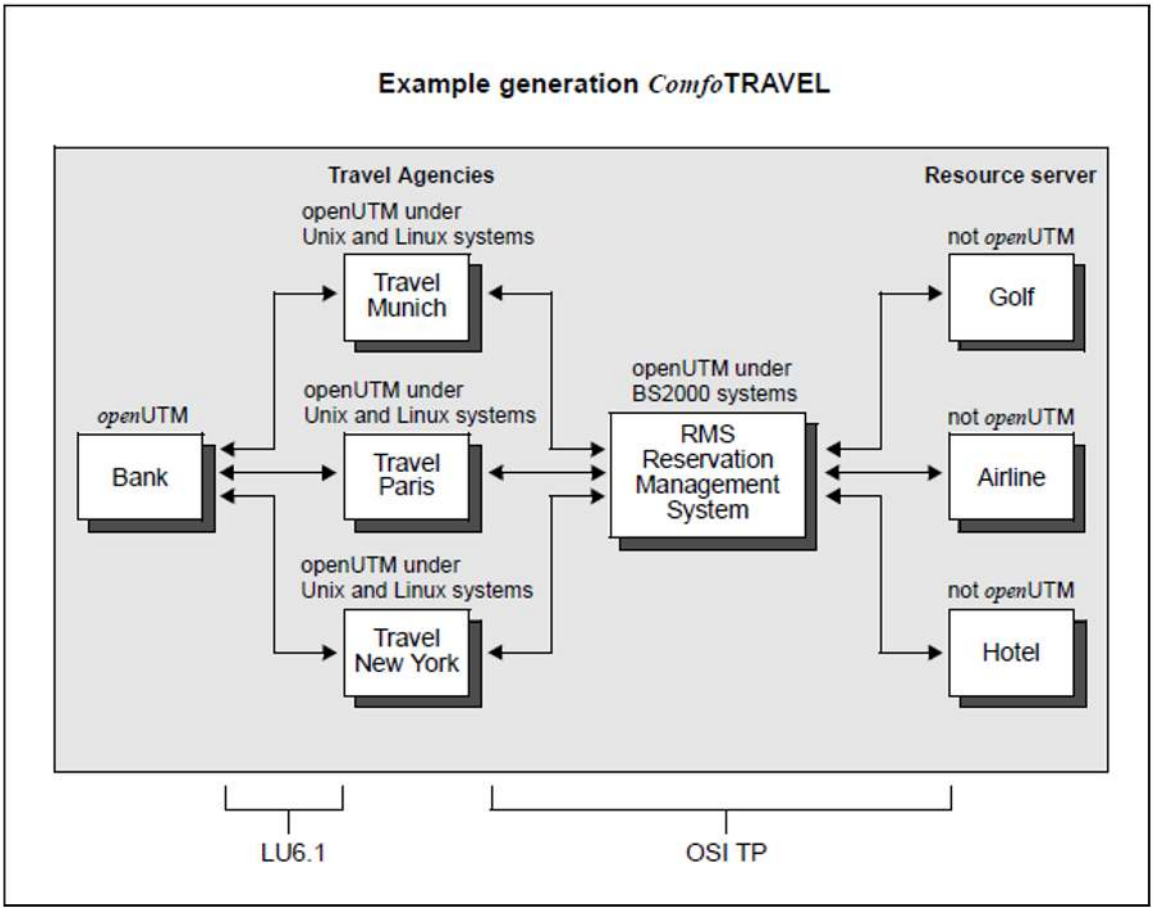


Figure 19: Example generation *Comfo*TRAVEL with the employed protocols

6.7.1 KDCDEF input file DYNAMIC.RMS for UTM-D application RMS

```
************************************
*       ADMINISTRATION programs     *
************************************
PROGRAM KDCADM, COMP=ILCS
PROGRAM KDCDADM,COMP=ILCS
PROGRAM KDCPADM,COMP=ILCS
************************************
*       RMS programs                *
************************************
PROGRAM AVALRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM RESRRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM CNCLRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM AUTHRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM INITRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM SHUTRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM ENQRRESP, COMP=ILCS, LOAD-MODULE=RMS
PROGRAM HNDLEXIT, COMP=ILCS, LOAD-MODULE=RMS
************************************
*       BOOKKEEPING programs        *
************************************
PROGRAM BOOKKEEP, COMP=ILCS, LOAD-MODULE=BOOKKEEP
************************************
*       PERSONNEL programs          *
************************************
PROGRAM PERSNNL, COMP=ILCS, LOAD-MODULE=PERSNNL
************************************
*       OFFICE programs             *
************************************
PROGRAM OFFICE,  COMP=ILCS, LOAD-MODULE=OFFICE
********************************************
*** TAC statements ***
********************************************
******* TAC  BOOKKEEP      *****
*
TAC BOOKKEEP, PROGRAM=BOOKKEEP, LOCK=7
******* TACS PERSNNL       *****
*
TAC OPERSNNL, PROGRAM=PERSNNL, LOCK=2
TAC MPERSNNL, PROGRAM=PERSNNL, LOCK=3
TAC CPERSNNL, PROGRAM=PERSNNL, LOCK=4
******* TACS OFFICE        *****
*
TAC OFFCHRG , PROGRAM=OFFICE , LOCK=5
TAC OFFADMIN, PROGRAM=OFFICE , LOCK=6
*
********* TACS RMS           *****
*
TAC AVALRESP, PROGRAM=AVALRESP, LOCK=8
TAC RESRRESP, PROGRAM=RESRRESP, LOCK=8
TAC CNCLRESP, PROGRAM=CNCLRESP, LOCK=8
TAC AUTHRESP, PROGRAM=AUTHRESP, LOCK=8
TAC INITRESP, PROGRAM=INITRESP, LOCK=8
TAC SHUTRESP, PROGRAM=SHUTRESP, LOCK=8
TAC ENQRRESP, PROGRAM=ENQRRESP, LOCK=8
*** ADMINISTRATION DIALOG ***
DEFAULT TAC ADMIN=Y, PROGRAM=KDCADM
```

```
TAC KDCTAC  , LOCK=1
TAC KDCLTERM, LOCK=1
TAC KDCPTERM, LOCK=1
TAC KDCSWTCH, LOCK=1
TAC KDCSEND , LOCK=1
TAC KDCAPPL , LOCK=1
TAC KDCUSER , LOCK=1
TAC KDCDIAG , LOCK=1
TAC KDCLOG  , LOCK=1
TAC KDCINF  , LOCK=1
TAC KDCHELP , LOCK=1
TAC KDCLPAP , LOCK=1
TAC KDCLTAC , LOCK=1
TAC KDCSHUT , LOCK=1
TAC KDCTCL  , LOCK=1
TAC TACDADM  , PROGRAM=KDCDADM, LOCK=1
TAC TACPADM  , PROGRAM=KDCPADM, LOCK=1
*** ADMINISTRATION ASYNCHRON ***
DEFAULT TAC TYPE=A
TAC KDCTACA , LOCK=1
TAC KDCLTRMA, LOCK=1
TAC KDCPTRMA, LOCK=1
TAC KDCSWCHA, LOCK=1
TAC KDCUSERA, LOCK=1
TAC KDCSENDA, LOCK=1
TAC KDCAPPLA, LOCK=1
TAC KDCDIAGA, LOCK=1
TAC KDCLOGA , LOCK=1
TAC KDCINFA , LOCK=1
TAC KDCHELPA, LOCK=1
TAC KDCLPAPA, LOCK=1
TAC KDCLTACA, LOCK=1
TAC KDCSHUTA, LOCK=1
TAC KDCTCLA , LOCK=1
*
*******************************************
*** USER statements                     ***
*******************************************
USER SUPERUSR, PERMIT=ADMIN,    PASS='$23ADM--', PROTECT-PW=(8, MAX) -
             , KSET=MASTER
USER UTMADMIN, PERMIT=ADMIN,    PASS='$23ADM--', PROTECT-PW=(8, MAX) -
             , KSET=UTMADMIN
USER CLERK   , FORMAT=*BOOK,    PASS='¤$*45jkl', PROTECT-PW=(8, MAX) -
             , KSET=BOOKKEEP
USER PRSNLMNG, FORMAT=*PERSNNL, PASS='78+lsd*/', PROTECT-PW=(8, MAX) -
             , KSET=MPRSNNL
USER MILLER  , FORMAT=*PERSNNL, PASS='7HGFKK*/', PROTECT-PW=( , MED) -
             , KSET=OPRSNNL
USER COMP    , FORMAT=*PERSNNL, PASS='7sdfKK*/', PROTECT-PW=( , MED) -
             , KSET=CPRSNNL
USER CHARGE  , FORMAT=*TRAVEL,  PASS='%aJ1df-+', PROTECT-PW=( , MED) -
             , KSET=OFFCHRG , LOCALE=(EN)
USER CHIEF   , FORMAT=*TRAVEL,  PASS='%aJs5f-+', PROTECT-PW=( , MED) -
             , KSET=OFFADMIN, LOCALE=(EN)
USER TPARIS  , FORMAT=*TRAVEL,  PASS='kj678+*',  PROTECT-PW=( , MED) -
             , KSET=TRVAGNCY, LOCALE=(FR, EU)
USER TNEWYORK, FORMAT=*TRAVEL,  PASS='56asdf$~', PROTECT-PW=( , MED) -
             , KSET=TRVAGNCY, LOCALE=(EN)
USER TMUNICH , FORMAT=*TRAVEL,  PASS='%as3f$0',  PROTECT-PW=( , MED) -
```

```
             , KSET=TRVAGNCY, LOCALE=(DE, EU)
USER TLONDON , FORMAT=*TRAVEL,  PASS='%4Jsdf-+', PROTECT-PW=( , MED) -
             , KSET=TRVAGNCY, LOCALE=(EN)
USER MANOFF  , FORMAT=*BOOK,    PASS='$23ADM--', PROTECT-PW=(8, MAX) -
             , KSET=OFFADMIN
*******************************************
*** PTERM/LTERM statements                ***
*******************************************
PTERM PRB22273, LTERM=PRINTER, PRONAM=PRO, PTYPE=T9021, CONNECT=YES
LTERM PRINTER, USAGE=O
PTERM RSO,     LTERM=RSO, PTYPE=*RSO, PRONAM=*RSO, CONNECT=YES
LTERM RSO,     USAGE=O
```

6.7.2 KDCDEF statements for UTM-D application RMS

```
*********************************************************************
*** K D C D E F - S T A T E M E N T S                             ***
*** FOR UTM-D-PROGRAM "RMS"                                       ***
*********************************************************************
ROOT RMSROOT
OPTION GEN=ALL
ACCOUNT ACC = YES
FORMSYS
MESSAGE MODULE = KCSMSGS, LOCALE=(EN)
MESSAGE MODULE = MSGSGER, LOCALE=(DE, EU)
MESSAGE MODULE = MSGSFRA, LOCALE=(FR, EU)
MAX LOCALE = (EN)
MAX KDCFILE = (RMS, DOUBLE)  -
   ,APPLINAME = APRMS        -
   ,APPLIMODE = S            -
   ,TASKS = 10               -
   ,ASYNTASKS = 3            -
   ,GSSBS = 200              -
   ,PGPOOL = 2048            -
   ,CACHESIZE = (512,50,RES) -
   ,CONN-USERS = 50          -
   ,RECBUF = (10,1024)       -
   ,KEYVALUE = 20            -
   ,LSSBS = 9                -
   ,LPUTBUF = 10             -
   ,LPUTLTH = 1948           -
   ,NRCONV = 1               -
   ,TERMWAIT = (600)         -
   ,DPUTLIMIT1 = (363,0,0,0) -
   ,DPUTLIMIT2 = (1,0,0,0)   -
   ,KB = 1024                -
   ,NB = 2048                -
   ,SPAB = 4096              -
   ,CLRCH = X'FF'
*********************************************************
*** RESERVE statement to allow dynamic administration ***
*********************************************************
RESERVE OBJECT=ALL
*********************************************************
*** DATABASE CONTROL statements                       ***
*********************************************************
DATABASE TYPE=UDS
DATABASE TYPE=XA, ENTRY=XAOSWD
*
*********************************************
*** SFUNC CONTROL statements              ***
*********************************************
SFUNC F1 , TAC = INITRESP
SFUNC F2 , TAC = SHUTRESP
SFUNC F5 , TAC = ENQRRESP
SFUNC F6 , TAC = AUTHRESP
SFUNC F7 , TAC = RESRRESP
SFUNC F8 , TAC = AVALRESP
SFUNC F20, TAC = CNCLRESP
*********************************************
*** KSET statements                       ***
```

```
*********************************************
KSET MASTER  , KEYS=MASTER        "SUPERUSER"
KSET UTMADMIN, KEYS=1             "Administrator of application"
KSET OPRSNNL , KEYS=2             "office personnel   / Büropersonal"
KSET MPRSNNL , KEYS=3             "personnel manager  / Personalchef"
KSET CPRSNNL , KEYS=4             "computer personnel / DV-Mitarbeiter"
KSET OFFCHRG , KEYS=5             "official in charge / Sachbearbeiter"
KSET OFFADMIN, KEYS=6             "administrator of office data"
KSET BOOKKEEP, KEYS=7             "book keeper"
KSET TRVAGNCY, KEYS=8             "travel agencies"
*********************************************
*** LOAD-MODULE statements                ***
*********************************************
LOAD-MODULE BOOKKEEP, VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
LOAD-MODULE PERSNNL , VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
LOAD-MODULE RMS     , VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
LOAD-MODULE OFFICE  , VERSION=@, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
*********************************************
*** EXIT statements                       ***
*********************************************
EXIT PROGRAM=HNDLEXIT, USAGE=START
EXIT PROGRAM=HNDLEXIT, USAGE=SHUT
*****************************************************
*** Read data which could be administered dynamically ***
*****************************************************
* use create-control-statements if application ran before
* CREATE-CONTROL-STATEMENTS *ALL, TO-FILE   = DYNAMIC.RMS.DATA -
*                               , FROM-FILE = COPIED.RMS.KDCA
OPTION DATA=DYNAMIC.RMS
*
*********************************************
*** TACCLASS statements                   ***
*********************************************
* not used
*********************************************
*** TLS statements                        ***
*********************************************
TLS TLSA
*********************************************
*** ULS statements                        ***
*********************************************
ULS ULSA
ULS ULSB
*********************************************
*** TPOOL statements                      ***
*********************************************
TPOOL LTERM=TP#,   NUMBER=100, PRONAM=*ANY, PTYPE=*ANY, KSET=MASTER
TPOOL LTERM=UPICR, NUMBER=100, PRONAM=*ANY, PTYPE=UPIC-R, KSET=MASTER
*********************************************
*** UTMD statements                       ***
*********************************************
UTMD MAXJR = 200, APT=(1,2,3,10), CONCTIME=25, PTCTIME=0
*********************************************
*** Generation of syntax                  ***
*********************************************
ABSTRACT-SYNTAX EUROSI, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12) -
                      , TRANSFER-SYNTAX = BER
*********************************************
* Generation of APPLICATION CONTEXTS       ***
```

```
*******************************************
*
* Without CCR
APPLICATION-CONTEXT EUROSIAC, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 12) -
                           , ABSTRACT-SYNTAX = (EUROSI)
*
* Include CCR
APPLICATION-CONTEXT EUOSICCR, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13) -
                           , ABSTRACT-SYNTAX = (EUROSI, CCR)
*
*****************************************************************
*** OSI TP generation                                         ***
*****************************************************************
*+-------------------------------------------------------------------+
*|            T   R   A   V   E   L   - Connections                  |
*+-------------------------------------------------------------------+
ACCESS-POINT RMS, T-SEL=C'RMS', S-SEL=('SRMS',ASCII) -
               , P-SEL=('PRMS',ASCII), AEQ=1
*
*       travel-agency MUNICH  <========>   RMS
OSI-CON MUNICH, LOCAL-ACCESS-POINT=RMS, OSI-LPAP=MUNICH                -
              , N-SEL=C'HOST0001', T-SEL=C'TRAV', S-SEL=(C'STRV',ASCII) -
              , P-SEL= (C'PTRV',ASCII)
*
*       travel-agency PARIS   <========>   RMS
OSI-CON PARIS , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=PARIS               -
              , N-SEL=C'ISO09', T-SEL=C'TRAV', S-SEL=(C'STRV',ASCII) -
              , P-SEL=(C'PTRV',ASCII)
*
*       travel-agency NEWYORK <========>   RMS
OSI-CON NEWYORK, LOCAL-ACCESS-POINT=RMS, OSI-LPAP=NEWYORK         -
               , N-SEL=C'ISO10', T-SEL=C'TRAV', S-SEL=('2',ASCII) -
               , P-SEL=('2',ASCII)
*
*       travel-agency LONDON  <========>   RMS
OSI-CON LONDON , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=LONDON          -
               , N-SEL=C'ISO06', T-SEL=C'TRAV', S-SEL=('2',ASCII) -
               , P-SEL=('2',ASCII)
*
OSI-LPAP MUNICH , ASS-NAMES=MUNICH, ASSOCIATIONS=4, CONNECT=0  -
                , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR      -
                , APT=(1,2,3,21),AEQ=1, KSET=TRVAGNCY
OSI-LPAP PARIS  , ASS-NAMES=PARIS, ASSOCIATIONS=4, CONNECT=0   -
                , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR      -
                , APT=(1,2,3,22), AEQ=1, KSET=TRVAGNCY
OSI-LPAP NEWYORK, ASS-NAMES=NEWYORK, ASSOCIATIONS=1, CONNECT=0 -
                , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR      -
                , APT=(1,2,3,23), AEQ=1, KSET=TRVAGNCY
OSI-LPAP LONDON , ASS-NAMES=LONDON, ASSOCIATIONS=1, CONNECT=0  -
                , CONTWIN=0, APPLICATION-CONTEXT=EUOSICCR      -
                , APT=(1,2,3,24), AEQ=1, KSET=TRVAGNCY
*
*+-------------------------------------------------------------------+
*|            From RMS to all servers                               |
*+-------------------------------------------------------------------+
*       RMS   <========> Server
*
OSI-LPAP BANK   , ASS-NAMES=BANK, ASSOCIATIONS=4, CONNECT=4   -
                , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR     -
```

```
                , APT=(1,2,3,30), AEQ=1
OSI-LPAP GOLF   , ASS-NAMES=GOLF, ASSOCIATIONS=4, CONNECT=4   -
                , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR     -
                , APT=(1,2,3,30), AEQ=2
OSI-LPAP HOTEL  , ASS-NAMES=HOTEL, ASSOCIATIONS=4, CONNECT=4  -
                , CONTWIN=4, APPLICATION-CONTEXT=EUOSICCR     -
                , APT=(1,2,3,30), AEQ=3
OSI-LPAP AIRLINE, ASS-NAMES=FLIGHT, ASSOCIATIONS=4, CONNECT=4 -
                , CONTWIN=4,APPLICATION-CONTEXT=EUOSICCR      -
                , APT=(1,2,3,30), AEQ=4
*
LTAC BANK, LPAP=BANK,    RTAC=BANK, STATUS=ON, TYPE=D
*
OSI-CON BANK   , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=BANK                  -
               , N-SEL=C'HOST0001', T-SEL=C'BANK', S-SEL=('SBNK',ASCII) -
               , P-SEL=(C'PBNK',ASCII)
OSI-CON GOLF   , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=GOLF                  -
               , N-SEL=C'HOST0001', T-SEL=C'GOLF', S-SEL=('SGLF',ASCII) -
               , P-SEL=('PGLF',ASCII)
OSI-CON HOTEL  , LOCAL-ACCESS-POINT=RMS, OSI-LPAP=HOTEL                 -
               , N-SEL=C'HOST0001', T-SEL = C'HOTL'                     -
               , S-SEL = ('SHTL',ASCII), P-SEL = ('PHTL',ASCII)
OSI-CON AIRLINE, LOCAL-ACCESS-POINT=RMS, OSI-LPAP=AIRLINE               -
               , N-SEL=C'HOST0001', T-SEL = C'FLGH'                     -
               , S-SEL=('SFLG',ASCII), P-SEL=('PFLG',ASCII)
END
```

6.7.3 KDCDEF input file DynamicTravel for UTM application TRAVEL

```
************************************
*    BANK program                  *
************************************
PROGRAM BANK,     COMP=C, SHARED-OBJECT=BANK
************************************
*    TRAVEL programs               *
************************************
PROGRAM SIGN1,    COMP=C
PROGRAM SIGN2,    COMP=C
PROGRAM BDTAC,    COMP=C
PROGRAM TRRECEIV, COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM STRTEX,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM MMENUE,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO1,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO2,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO3,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO4,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO5,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINFO6,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM CANCEL,   COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM CANCALL,  COMP=C, SHARED-OBJECT=TRAVEL
PROGRAM TRINQU,   COMP=C, SHARED-OBJECT=TRAVEL
**** Administration
PROGRAM KDCADM,   COMP=C
PROGRAM KDCDADM,  COMP=C
PROGRAM KDCPADM,  COMP=C
************************************
*      TACS BANK                   *
************************************
TAC BANK, PROGRAM=BANK, LOCK=5
************************************
*      TACS TRAVEL AGENCY          *
************************************
TAC KDCBADTC, PROGRAM=BDTAC, TYPE=D
TAC KDCSGNTC, PROGRAM=SIGN1, TYPE=D
TAC SIGNON2 , PROGRAM=SIGN2, TYPE=D
TAC MMENUE  , PROGRAM=MMENUE  , LOCK=5
TAC INFO1   , PROGRAM=TRINFO1 , LOCK=5
TAC INFO2   , PROGRAM=TRINFO2 , LOCK=5
TAC INFO3   , PROGRAM=TRINFO3 , LOCK=5
TAC INFO4   , PROGRAM=TRINFO4 , LOCK=5
TAC INFO5   , PROGRAM=TRINFO5 , LOCK=5
TAC INFO6   , PROGRAM=TRINFO6 , LOCK=5
TAC TRRECEIV, PROGRAM=TRRECEIV, LOCK=5
TAC CANCEL  , PROGRAM=CANCEL  , LOCK=5, CALL=NEXT,  TYPE=D
TAC CANCALL , PROGRAM=CANCALL , LOCK=5, CALL=FIRST, TYPE=D
TAC INQUIRY , PROGRAM=TRINQU  , LOCK=5
*
**** ADMINISTRATION DIALOG ***
TAC KDCTAC  , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLTERM, LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCPTERM, LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCSWTCH, LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCSEND , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCAPPL , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCUSER , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
```

```
TAC KDCDIAG , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLOG  , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCINF  , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCHELP , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLPAP , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCLTAC , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCSHUT , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC KDCTCL  , LOCK=1, ADMIN=Y, PROGRAM=KDCADM
TAC TACDADM  , PROGRAM=KDCDADM, LOCK=1, ADMIN=Y
TAC TACPADM  , PROGRAM=KDCPADM, LOCK=1, ADMIN=Y
*** ADMINISTRATION ASYNCHRON ***
TAC KDCTACA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLTRMA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCPTRMA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCSWCHA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCUSERA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCSENDA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCAPPLA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCDIAGA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLOGA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCINFA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCHELPA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLPAPA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCLTACA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCSHUTA, LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
TAC KDCTCLA , LOCK=1, ADMIN=Y, TYPE=A, PROGRAM=KDCADM
*
*********************************************
*** USER statements                       ***
*********************************************
USER SUPERUSR, PERMIT=ADMIN,    PASS='$23ADM--', PROTECT-PW=(8, MAX)        -
             , KSET=MASTER
USER UTMADMIN, PERMIT=ADMIN,    PASS='$23ADM--', PROTECT-PW=(8, MAX)        -
             , KSET=UTMADMIN
USER CHARGE1 , PASS='%aJ1df-+', PROTECT-PW=( ,MED)  -
             , KSET=OFFCHRG
USER CHARGE2 , PASS='%aJ1df-+', PROTECT-PW=( , MED) -
             , KSET=OFFCHRG
*            .
*********************************************
*** PTERM/LTERM statements                ***
*********************************************
PTERM PRINTX,  LTERM=PRINTER, PTYPE=PRINTER, CONNECT=YES
LTERM PRINTER, USAGE=O
```

6.7.4 KDCDEF statements for UTM application TRAVEL

```
*********************************************************************
*** K D C D E F - S T A T E M E N T S                             ***
*** FOR UTM-PROGRAM "TRAVEL"                                      ***
*********************************************************************
ROOT TRAVROOT
OPTION GEN=ALL
FORMSYS
MESSAGE MODULE = KCSMSGS
MAX KDCFILE = (TRAVFILE, DOUBLE) -
   ,APPLINAME = APTRAVEL         -
   ,APPLIMODE = S                -
   ,TASKS = 7                    -
   ,ASYNTASKS = 3                -
   ,GSSBS = 200                  -
   ,PGPOOL = (2048)              -
   ,CACHESIZE = (512,50)         -
   ,CONN-USERS = 50              -
   ,TRACEREC = 30000             -
   ,RECBUF = (10,1024)           -
   ,KEYVALUE = 20                -
   ,LSSBS = 9                    -
   ,LPUTBUF = 10                 -
   ,LPUTLTH = 1948               -
   ,NRCONV = 1                   -
   ,TERMWAIT = (600)             -
   ,DPUTLIMIT1 = (363,0,0,0)     -
   ,DPUTLIMIT2 = (1,0,0,0)       -
   ,KB = 1024                    -
   ,NB = 2048                    -
   ,SPAB = 4096                  -
   ,CLRCH = X'FF'                -
   ,SEMARRAY    =(00001221,5)    -
   ,IPCSHMKEY   = 00012210       -
   ,KAASHMKEY   = 00012220       -
   ,CACHESHMKEY = 00012230       -
   ,OSISHMKEY   = 00012244       -
   ,XAPTPSHMKEY = 00012254
*******************************************************
*** Read data which can be administrated dynamically ***
*******************************************************
* if application ran before use create-control-statements
* CREATE-CONTROL-STATEMENTS *ALL, TO-FILE   = dynamicTravel      -
*                             , FROM-FILE = TRAVFILE/copied.KDCA
OPTION DATA=DynamicTravel
*
*******************************************************
*** RESERVE statement to allow dynamic administration ***
*******************************************************
RESERVE OBJECT=ALL
*******************************************************
***    RMXA                                          ***
*******************************************************
RMXA XASWITCH=xaoswd
*********************************************
*** SHARED-OBJECT statements               ***
*********************************************
```

```
SHARED-OBJECT TRAVEL, LIB=DYNPROGLIB, LOAD-MODE=STARTUP
SHARED-OBJECT BANK,   LIB=DYNPROGLIB, LOAD-MODE=STARTUP
*******************************************
*** KSET statements                       ***
*******************************************
KSET MASTER  , KEYS=MASTER        "SUPERUSER"
KSET UTMADMIN, KEYS=1             "Administrator of application"
KSET OFFCHRG , KEYS=5             "official in charge / Sachbearbeiter"
*******************************************
*** TPOOL statements                      ***
*******************************************
TPOOL LTERM=TP#,   NUMBER=100, PTYPE=TTY, KSET=MASTER
TPOOL LTERM=UPICR, NUMBER=100, PTYPE=UPIC-R, KSET=MASTER
*******************************************
*** Generation of syntax                  ***
*******************************************
ABSTRACT-SYNTAX EUROSI, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 3, 12) -
, TRANSFER-SYNTAX = BER
*******************************************
* Generation of APPLICATION CONTEXTS       ***
*******************************************
*
* Without CCR
APPLICATION-CONTEXT EUROSIAC, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 12) -
                           , ABSTRACT-SYNTAX = (EUROSI)
*
* Include CCR
APPLICATION-CONTEXT EUOSICCR, OBJECT-IDENTIFIER = (1, 3, 9990, 1, 4, 13) -
                           , ABSTRACT-SYNTAX = (EUROSI, CCR)
*
*****************************************************************
*** OSI TP generation                                         ***
*****************************************************************
*******************************************
*** UTMD statements                       ***
*******************************************
UTMD MAXJR = 200, APT=(1,2,3,21), CONCTIME=25, PTCTIME=0
*+-------------------------------------------------------------+
*|                     R   M   S  -   Connections              |
*+-------------------------------------------------------------+
ACCESS-POINT TRAVEL, T-SEL=C'TRAV', S-SEL= (C'STRV',ASCII) -
                   , P-SEL= (C'PTRV',ASCII), AEQ=1 -
                   , LISTENER-PORT=30003, T-PROT=RFC1006 , TSEL-FORMAT=T
*+-------------------------------------------------------------+
*|            From travel agency to RMS                        |
*+-------------------------------------------------------------+
*       travel-agency   <========>   RMS
OSI-CON RMS , LOCAL-ACCESS-POINT=TRAVEL, OSI-LPAP=RMS,N-SEL=C'HOST0001'-
            ,T-SEL=C'RMS',S-SEL= (C'SRMS',ASCII), P-SEL= (C'PRMS',ASCII)-
            ,LISTENER-PORT=102, T-PROT=RFC1006, TSEL-FORMAT=T
OSI-LPAP RMS, ASS-NAMES=RMS, ASSOCIATIONS=4, CONNECT=0, CONTWIN=4      -
            , APPLICATION-CONTEXT=EUOSICCR, APT=(1,2,3,10),AEQ=1
*
*+-------------------------------------------------------------+
*|                     B A N K   -   Connections               |
*+-------------------------------------------------------------+
SESCHA  PLUC, PLU=Y, PACCNT=0, CONNECT=Y
LPAP LPBANK, SESCHA=PLUC
BCAMAPPL SMP30041                                    -
```

```
        ,T-PROT=RFC1006                                    -
        ,LISTENER-PORT=30004,TSEL-FORMAT=T
* Connection 1 for sending ---> BANK---------------*
CON SMP30114,PRONAM=local,BCAMAPPL=SMP30041,LPAP=LPBANK    -
            ,T-PROT=RFC1006                                -
            ,LISTENER-PORT=30001,TSEL-FORMAT=T
LSES SMP30141,RSES=SMP30141,LPAP=LPBANK
* Connection 2 for sending    ---> BANK---------------*
CON SMP30214,PRONAM=local,BCAMAPPL=SMP30041,LPAP=LPBANK    -
            ,T-PROT=RFC1006                                -
            ,LISTENER-PORT=30001,TSEL-FORMAT=T
LSES SMP30241,RSES=SMP30241,LPAP=LPBANK
* Connection 3 for sending    ---> BANK---------------*
CON SMP30314,PRONAM=local,BCAMAPPL=SMP30041,LPAP=LPBANK    -
            ,T-PROT=RFC1006                                -
            ,LISTENER-PORT=30001,TSEL-FORMAT=T
LSES SMP30341,RSES=SMP30341,LPAP=LPBANK
*-----------------------------------------------------------------*
*        LTAC's              -------> BANK
*-----------------------------------------------------------------*
LTAC  bank, RTAC=BANK, WAITTIME=(10,30), LPAP=LPBANK
*-----------------------------------------------------------------*
*        LTAC's              -------> RMS
*-----------------------------------------------------------------*
LTAC AVALRESP, LPAP=RMS
LTAC RESRRESP, LPAP=RMS
LTAC CNCLRESP, LPAP=RMS
LTAC AUTHRESP, LPAP=RMS
LTAC INITRESP, LPAP=RMS
LTAC SHUTRESP, LPAP=RMS
LTAC ENQRRESP, LPAP=RMS
*
END
```

6.8 KDCDEF messages

The KDCDEF generation tool logs the defined parameters on SYSLST (BS2000 systems) or on *stdout* (Unix, Linux and Windows systems). It also outputs UTM messages relating to execution of the program, with message numbers ranging from K400 to K549. Apart from UTM messages K401, K513 and K514 all KDCDEF messages are output both to SYSLST and to SYSOUT or to *stderr* and to *stdout*. UTM messages K401, K513 and K514 is output only to SYSOUT or *stderr*.

On Unix and Linux systems KDCDEF uses NLS message catalogs to output messages.

UTM messages relating to incorrect statements are preceded by the number of the incorrect statement. The openUTM manual ”Messages, Debugging and Diagnostics” lists all UTM messages together with information on the corrective actions to be performed.

7 Changing the configuration of an application dynamically

This chapter describes what to note in the KDCDEF generation of the application if you want to use the dynamic configuration functions in your application. On the program interface, openUTM provides KDCADMI functions as well as functions available at the administration workstation WinAdmin or the web application WebAdmin with which you can enter objects in the configuration of the application or delete them from the configuration while the application is running. This increases the availability of UTM applications, because a regeneration of the application with KDCDEF, which necessitates an interruption to operation, is required much less often. In order to use the functions for dynamic configuration, you must reserve table locations in the object tables of openUTM when generating with the KDCDEF control statement RESERVE.

This means that services as well as clients and printers can be entered dynamically in the configuration with the assigned LTERM partners, and also means that user IDs can be created dynamically. All of these objects can also be deleted dynamically.

You can dynamically create and delete the following objects:

- transaction codes and TAC queues
- program units and VORGANG exits (only in applications with load modules, shared objects or DLLs)
- user IDs
- LTERM partners
- key sets
- local service names
- transport connections to LU6.1 partner applications and LU6.1 session names
- communication partners that are TS applications, UPIC clients or terminals
- printers.

To be able to use the functions of dynamic configuration, you must create administration programs or use the openUTM components WinAdmin or WebAdmin. By calling KC_CREATE_OBJECT on the program interface for administration you can enter new objects in the configuration, and by calling KC_DELETE_OBJECT you can delete objects from the configuration. The openUTM manual “Administering Applications” describes what to note when creating administration programs for dynamically entering objects and when deleting objects from the configuration of the application.

> The dynamic configuration functions can also be used in full in the function variant UTM-F. openUTM logs all changes to the configuration in the KDCFILE. The modified configuration data then also remains available for the next application run, as with UTM-S.

To allow you to incorporate objects into the configuration of your UTM application dynamically, you must make certain preparations (see "Reserving locations in the KDCFILE object tables" and "Prerequisites for entering objects dynamically") when generating the application with KDCDEF.

No preparations are necessary in the KDCDEF generation for deleting objects from the configuration.

7.1 Reserving locations in the KDCFILE object tables

The configuration data of a UTM application is stored in the object tables of the KDCFILE, which is created in the KDCDEF generation of the application. These object tables can only be expanded dynamically insofar as free table locations are available. For this reason, free table locations for objects you do not want to incorporate into the application configuration until the application is running, must still be reserved when generating with the KDCDEF statement RESERVE. You can reserve table locations for the following UTM objects:

UTM object	**Object type**
User IDs	USER
TS applications, UPIC clients, terminals and printers	PTERM
LTERM partners	LTERM
Program units and VORGANG exits	PROGRAM
Transaction codes and TAC queues	TAC
Transport connections to LU6.1 partner applications	CON
LU6.1 session names	LSES
Key sets	KSET
Local service names for remote applications	LTAC

With the RESERVE statement in section "RESERVE - reserve table locations for UTM objects", you define the number of empty table locations to be created for an object type; this corresponds to the number of individual objects of the respective object type that can be configured dynamically.

The number of table locations that can be created for each object type is limited by the number of names that can be generated. See also the table in section "Number of names".

The empty table locations in the object tables are reserved for specific object types, i.e. a table location you reserved for an LTERM partner for example, cannot be occupied by a transaction code, etc.

During the application run, the number of objects of a particular type that can be configured dynamically corresponds to the number of empty table locations you reserved with KDCDEF. Deleting another object of the same type does not immediately release a table location for a new object.

User IDs are one exception to this. You can delete user IDs in one of two ways, "delayed" or "immediately". If a user ID is deleted with "delayed", then the table locations remain reserved (as for the other types of objects). If the user ID is removed "immediately" from the configuration, the table location for this user ID is released and can be used immediately for a new user ID.

Please note the following when reserving table locations with RESERVE: openUTM internally creates a user ID for each UPIC client and each TS application entered dynamically in the configuration. Therefore, in UTM applications generated with user IDs (the KDCDEF generation contains at least one USER statement), an additional table location for user IDs must be reserved for each client of type APPLI, SOCKET, UPIC-R or UPIC-L to be entered dynamically. These table locations are not released when the clients are deleted (corresponds to a “delayed“ delete). In applications without user IDs, these table locations are reserved internally by openUTM.

Examples

```
RESERVE OBJECT=LTERM, NUMBER=100
```

This means that up to 100 LTERM partners can be entered dynamically in the configuration.

```
RESERVE OBJECT=LTERM, PERCENT=200
```

In this case, the number of reserved table locations was defined relative to the number of statically generated LTERM partners. Twice as many (200%) LTERM partners can be created dynamically as were entered statically in the KDCDEF generation. If 50 LTERM partners were entered in the KDCDEF generation, another 100 LTERM partners can be entered dynamically.

```
RESERVE OBJECT=ALL, NUMBER=100
```

This means that 100 objects can be entered dynamically for each object type, i.e. 100 user IDs, 100 LTERM partners, etc.

```
RESERVE OBJECT=USER, NUMBER=0
```

This statement means that the number of objects of the specified type (here USER) can be increased dynamically up to the maximum value that can be generated.

Due to the large amount of space required by the tables, it is advisable to specify a value != 0 for NUMBER in order to reduce the space requirement of the application.

7.2 Prerequisites for entering objects dynamically

This section describes what objects you must generate statically beforehand and what prerequisites must be met before you can dynamically enter program units, VORGANG exits, transaction codes, user IDs and LU6.1 connections.

Note that the statically generated limit values also apply for dynamically generated objects, e.g. the value defined with MAX ...,KEYVALUE= applies to dynamically generated key sets.

Generating lock codes, BCAMAPPL names, formatting system and LPAP partners

The following objects must be statically generated in KDCDEF:

- The lock codes you want to assign to transaction codes and LTERM partners as data access control must lie within the range between 1 and the maximum value defined with MAX,...KEYVALUE= *number*. At the same time, you must generate the key sets that contain the key codes corresponding to the lock codes.

  The lock/key code concept is described in detail in the openUTM manual “Concepts und Functions”.
- All names of the local application (BCAMAPPL names) via which the connections are to be established to clients or printers, must be generated with KDCDEF. Remember especially that a separate BCAMAPPL name must be generated for the connection of TS applications via the socket interface (PTYPE=SOCKET).
- If start formats are to be assigned to user IDs and LTERM partners, a format handling system must be generated with the FORMSYS statement in the KDCDEF generation. If #formats are used as start formats, a sign-on service must also be generated.
- If you want to enter LU6.1 connections or session names dynamically, the LPAP partners and the session characteristics (SESCHA statement) must be statically generated.

Prerequisites for program units and VORGANG exits

New program units and VORGANG exits can only be incorporated dynamically into the configuration of the application if the UTM application

- *BS2000 systems:*

  The application was generated with load modules (KDCDEF generation with LOAD-MODULE statements), and the functionality of the BLS must be used for linking and loading the application program.

  The new program unit must be linked in a load module which was defined in the KDCDEF generation. This load module must not be linked statically in the application program (LOAD-MODE=STATIC), because this type of load module cannot be exchanged dynamically.
- *Unix , Linux and Windows systems:*

  The application was generated with shared objects or DLLs (KDCDEF generation with SHARED-OBJECT statements).
  The new program unit must be linked in a shared object or DLL which was defined in the KDCDEF generation.

At least one program unit must be generated statically with KDCDEF for each programming language in which you want to create program units of your application. Only then the language connection modules and runtime systems are required for operation contained in the application program.

BS2000 systems:
For program units compiled with ILCS-capable compilers (COMP=ILCS), it is sufficient to generate one program unit with COMP=ILCS in the KDCDEF generation. It is not necessary to issue PROGRAM statements for the various programming languages.

Prerequisites for transaction codes

Please note the following for the dynamic configuration of transaction codes:

- Transaction codes for program units that use an X/Open program interface can only be created dynamically if at least one transaction code has been configured for an X/Open program unit in the KDCDEF generation (TAC statement with API!=KDCS).
- If you want to divide the transaction codes into TAC classes to control job processing, then you must create at least one TAC class in the KDCDEF generation. TAC classes can be created in two ways in the KDCDEF generation:
  - You generate a transaction code, and you specify a TAC class in the TACCLASS operand (TAC statement) that is then implicitly created by KDCDEF.
  - If you run the application **without** priority control (the application does not contain any TAC-PRIORITIES statements), then you can create the TAC classes by writing a TACCLASS statement.

  If you created a TAC class in the KDCDEF generation, then the transaction codes you enter dynamically can be assigned to any TAC class between 1 and 16. The TAC classes are then created implicitly by openUTM. These TAC classes can be administered.

  If the application is generated **without** TAC-PRIORITIES, then openUTM assigns the process numbers (TASKS) for implicitly created TAC classes as follows: 1 for dialog TAC classes (classes 1 through 8)
  and 0 for asynchronous TAC classes (classes 9 through 16).

  Asynchronous TAC classes (9 through 16) are only created by openUTM, however, if you have set ASYNTASKS > 0 in the MAX statement in the generation.
- In applications with TAC classes **without** priority control, you can only dynamically create transaction codes that start the program unit runs with blocking calls when TAC classes have been statically generated with PGWT=YES (dialog and/or asynchronous TAC classes).
  Dialog and asynchronous TAC classes with PGWT=YES must therefore be generated explicitly in the KDCDEF generation with TACCLASS statements.
  You must also set MAX TASKS-IN-PGWT > 0.
- In applications **with** priority control (with TAC-PRIORITIES statements) , you can only dynamically create transaction codes that start the program unit runs with blocking calls (TAC ...,PGWT=YES) when MAX TASKS-IN-PGWT>0 was set in the KDCDEF generation.

Prerequisites for user IDs

User IDs can only be entered dynamically if your application is generated with user IDs. In this case, your KDCDEF generation must contain at least one USER statement. At least one user ID must be configured with administration authorization, so that the calls for dynamic administration can be executed under this user ID.

If user IDs with ID cards are also to be configurable, the percentage of table locations that can be occupied by user IDs with ID card must be explicitly specified when reserving the table locations with the RESERVE statement.

You must generate the length of the ID card information statically in the KDCDEF generation using the MAX statement for user IDs with ID cards:
MAX...,CARDLTH=*length*

8 The tool KDCUPD - updating the KDCFILE

You can use the KDCUPD tool after regenerating your UTM application to transfer important user data and administration information of the production application from the old KDCFILE to the new one. In addition, you can use the KDCUPD update tool to switch from an older openUTM version to the current new openUTM version without losing the data from the previous production application in the KDCFILE.

The same applies to UTM cluster applications on Unix, Linux and Windows systems, both for the KDCFILEs of the node applications and for the user data and management information in the UTM cluster files created during generation.

You can use the KDCUPD statement TRANSFER to control which data are to be transferred. KDCUPD automatically carries out a consistency check for the KDCFILE files prior to transfer.

You can use the KDCUPD statement CHECK to check the completeness and consistency of the KDCFILE files of an application without transferring data.

> **i** In the case of UTM cluster applications, a distinction is made in the KDCUPD run depending on whether the data of a KDCFILE or the data from the UTM cluster files is to be transferred. For details, see "Update generation for UTM cluster applications".

8.1 Overview

This section provides an overview of

- Version upgrades
- Prerequisites
- Data backups
- Scope of transfer, i.e. what data is transferred

8.1.1 Supported upgrades

You can use the KDCUPD utility program to transfer data from applications from openUTM versions 6.3, 6.4, 6.5 and 7.0.

The KDCUPD utility program from openUTM V7.0 also supports the following upgrades:

openUTM V6.3 -> openUTM V7.0
openUTM V6.4 -> openUTM V7.0
openUTM V6.5 -> openUTM V7.0
openUTM V7.0 -> openUTM V7.0

On Unix, Linux and Windows systems, KDCUPD in V7.0 also supports a transfer from 32-bit to 64-bit architecture, see chapter "Update generation with transfer from 32-bit to 64-bit architecture".

When changing versions, you must create a new KDCFILE with the KDCDEF of the new openUTM version.

KDCUPD does not support a change from a newer openUTM to an older one.

For UTM applications on Unix , Linux and Windows systems applies:

An upgrade from a standalone UTM application to a UTM cluster application or vice versa is only possible within Version 7.0. If you want to change from a standalone UTM application of an previous version to a V7.0 UTM cluster application then you must first convert the standalone UTM application from the previous version to V7.0 before you can convert it to a cluster application in V7.0.

8.1.2 Prerequisite for using KDCUPD

The **prerequisites** for running KDCUPD are:

- You have used the KDCDEF to create a new KDCFILE.

  If the application is a UTM cluster application and if a cluster update is to performed then it is necessary to use the KDCDEF generation tool to create new UTM cluster files as well as the new KDCFILE.
- The application was terminated normally (e.g. with the administration command KDCSHUT N, W or G). If the application was terminated abnormally, then you must execute a warm start beforehand and then terminate the application normally. In the case of a cluster update, all the node applications must have been terminated normally.

  In the case of a cluster update, all the node applications must have been terminated normally.

8.1.3 Backing up data

Before you start your work, please read the following notes:

> **CAUTION!**
>
> - When you create the new KDCFILE with KDCDEF, you must ensure that you **do not accidentally overwrite the old KDCFILE** and thus destroy important application data!
> - The records in user log files (USLOG files) must be saved before the application is restarted because openUTM overwrites the current USLOG file generation from the beginning after a KDCUPD run.

There are a number of ways of avoiding overwriting the KDCFILE:

- As shown here, you create the new KDCFILE before terminating the application. You use the same base name, but set up the KDCFILE under another ID (BS2000 systems) or in another directory (Unix, Linux and Windows systems). After you terminate the application, you must rename or copy the files for the subsequent KDCUPD run.
- First terminate the application and then rename the old KDCFILE and all associated files by changing the base name. Alternatively, copy the old KDCFILE and all associated files to a different ID (BS2000 systems) or to a different directory (Unix, Linux and Windows systems). Then start the KDCDEF run to generate the new KDCFILE with the same base name.
  Specify the following in the subsequent KDCUPD run:
  - KDCFILE OLD= *base_name-renamed/copied-KDCFILE*
  - KDCFILE NEW= *base_name-new-KDCFILE*
- Use a new base name for the new KDCFILE and work with this name in the KDCUPD run. When you subsequently start the application, you can either use the new base name or continue to use the previous base name after copying and renaming the files.

8.1.4 What data does KDCUPD transfer?

This section lists the data that is transferred, indicates the dependencies of the UTM variants and generation parameters and describes in greater detail which user data is always transferred and which it might sometimes not be possible to transfer.

Transfer in standalone applications

The data that KDCUPD transfers from the old KDCFILE to the new one depends on the variant of the UTM application, see also section "Transfer of user data":

- UTM-F applications

  KDCUPD transfers certain changes to the administration data:

  - Passwords and RSA keys
  - if data compression is permitted by means of the generation: information whether data compression is enabled.
  - locales of users and version numbers of load modules on BS2000 systems
  - version number of shared objects on Unix and Linux systems
  - version number of DLLs on Windows systems.

  All available RSA keys of levels 1 to 4 are also transferred in a KDCUPD run. Active keys and keys created using administration facilities but not yet activated are transferred. If, in the old KDCFILE, there are no RSA keys in an encryption level, then nothing is transferred for this level. It can therefore happen that RSA keys generated for this encryption level in the new KDCFILE are not overwritten with 0.

- UTM-S applications

  For UTM-S applications, KDCUPD transfers all changes as for UTM-F applications and in addition transfers administration data and current user data such as global secondary storage areas, asynchronous messages, TLS or ULS areas, and service-specific information etc. from the previous KDCFILE to a newly generated KDCFILE. In the data transfer, the KDCUPD checks whether the owner, the destination or the initiator of the data is missing in the new KDCFILE or if it was deleted by the administration in the previous application run. In this case, KDCUPD does not transfer the data and logs this event.

Transfer in UTM cluster applications on Unix, Linux and Windows systems

In UTM cluster applications, the scope of the transferred data also depends on whether you are performing a node update or a cluster update.

Cluster update

When a cluster update is performed in a UTM cluster application, the management and user data for the GSSB, ULS and the service-specific information from the previous UTM cluster files is imported into the new UTM cluster files irrespective of the variant of the UTM application. If data cannot be transferred, for example because the owner of the service specific data is not present in the new UTM cluster files, then this is logged.

Node update

In the case of a node update, the data that KDCUPD transfers from the old to the new KDCFILE depends on the variant of the UTM application:

- UTM-F applications

  KDCUPD transfers certain changes to the administration data:

  - RSA keys
  - if data compression is permitted by means of the generation: information whether data compression is enabled.
  - version number of shared objects on Unix and Linux systems
  - version number of DLLs on Windows systems.

  All available RSA keys of levels 1 to 4 are transferred in a KDCUPD run. Active keys and keys created using administration facilities but not yet activated are transferred. If, in the old KDCFILE, there are no RSA keys in an encryption level, then nothing is transferred for this level. It can therefore happen that RSA keys generated for this encryption level in the new KDCFILE are not overwritten with 0.

- UTM-S applications

  For UTM-S applications, KDCUPD transfers all changes as for UTM-F applications and in addition transfers administration data and current user data such as asynchronous messages, TLS areas from the previous KDCFILE to a newly generated KDCFILE. In the data transfer, the KDCUPD checks whether the owner, the destination or the initiator of the data is missing in the new KDCFILE or if it was deleted by the administration in the previous application run. In this case, KDCUPD does not transfer the data and logs this event.

For further details, see "Update generation for UTM cluster applications". The effect of the individual parameters in node updates and cluster updates can be found in the description of the TRANSFER statement, see "TRANSFER - control the data transfer of the user data".

8.1.4.1 Changing generation parameters

KDCUPD compares the generations of the two KDCFILEs. Depending on the results of these checks, KDCUPD cannot transfer some items of data and in some cases must reject transfer completely.

If the KDCUPD run detects that the database configurations of the two generations are incompatible then an error message is output and the KDCUPD run is terminated abnormally.

No transfer on BS2000 systems

When generating new KDCFILE the user is basically permitted to change all the generation parameters compared with the old KDCFILE. However, there are some exceptions which apply only to UTM-S applications:

- The entire transfer is denied by KDCUPD fif Old KDCFILE generated with formatting, new KDCFILE without because running an application with the KDCFILE would lead to errors.
- If the database generations have changed in terms of the number and sequence of the databases, no open services are transferred.
  Exception:
  If more databases are generated in the new generation than in the old one and the existing sequence from the old generation is retained, everything is transferred.
- In case of a version change the entire transfer is rejected, if the database generations have changed in terms of the number and sequence of the databases.

In the case of the UTM-F variant, differences of this kind do not prevent KDCUPD from carrying out the transfer.

Limited transfer

There are generation differences between the old and new KDCFILE which in principle permit transfer, but for which individual messages or data areas cannot be transferred. KDCUPD logs events such as these to SYSOUT or SYSLST (on BS2000 systems) or to stdout or stderr (on Unix, Linux and Windows systems) and continues transfer to the new KDCFILE.

Examples

> If an LTERM partner is no longer defined in the new KDCFILE, KDCUPD cannot transfer any FPUT messages and TLS areas for these LTERM partners.
>
> If the communication area of a dialog service is larger than the maximum communication area length (operand MAX KB=...) in the new KDCFILE, KDCUPD rejects transfer of this service.

In general, the generation values of the new KDCFILE apply when transferring with KDCUPD.

8.1.4.2 Transfer of user data

The transfer of user data can be controlled using the KDCUPD TRANSFER statement.

Please note that in the case of standalone applications as well as of node updates in UTM cluster applications, the following remarks only apply to UTM-S.

User data always transferred by KDCUPD

KDCUPD always transfers the following data of a KDCFILE, irrespective of the specifications in the TRANSFER control statement:

- asynchronous messages in USER queues when the user, generating LTERM, and generating USER also exist in the new KDCFILE

User data which KDCUPD transfers optionally

The TRANSFER control statement allows you to control which of the following data KDCUPD is to transfer to the new KDCFILE or the UTM cluster files:

- Dialog services started by a terminal or a TS application of type APPLI.
- Dialog services started by a UPIC client.
- Dialog services started by a TS application of type SOCKET.
- Passwords for the user IDs and - if generated - the validity period remaining, minimum wait time until the next password change, and the password history
- Secondary storage area GSSB, TLS, and ULS
- Queued output jobs
- Queued messages to local asynchronous services and TAC queues as well as open asynchronous services
- Queued messages to local partners
- Queued messages to remote partners
- Queued messages to temporary queues
- Current version number of the loadable objects (load objects on BS2000 systems, shared objects on Unix and Linux systems, DLLs on Windows systems)
- The locales of users (only on BS2000 systems)

When services are transferred, all service-specific data is transferred:

- Local secondary storage area data
- Saved dialog messages
- Communication areas
- Batch stacked services (only for standalone applications)

Data not transferred by KDCUPD

The following is not transferred by KDCUPD:

- Objects which have been added with the administration function such as new USERs.
- Data belonging to open, distributed services.
  KDCUPD does not issue a message that this data was not transferred!

- Open dialog services of users when the user does not exist in the new KDCFILE.
- Open asynchronous services when the user who started the service or the LTERM or (OSI-)LPAP partner from which the asynchronous service was started does not exist in the new KDCFILE.
- Queued messages when the destination of the message, the user who generated the message, or the LTERM or (OSI-)LPAP partner from which the asynchronous service was started does not exist in the new KDCFILE.
- The TLS or ULS storage areas when the corresponding LTERM, (OSI-)LPAP, the associated USER, or the associated session or association does not exist in the new KDCFILE.
- Service stacks in UTM cluster applications.

8.2 Update generation for standalone UTM applications

Steps

> ! Before performing the operations below, please read section "Backing up data"!

Steps 1 to 5 on the following pages explain how to use the tool KDCUPD to update the KDCFILE.

1. Create the new KDCFILE with KDCDEF
2. Terminate the application normally
3. Rename/copy the old KDCFILE and back up the user log file
4. Call KDCUPD
5. Start application

These steps are described in detail on the following pages. The method shown here is only one of several possibilities. In all cases, you must ensure that the KDCFILE is not overwritten, see warning in section "Backing up data".

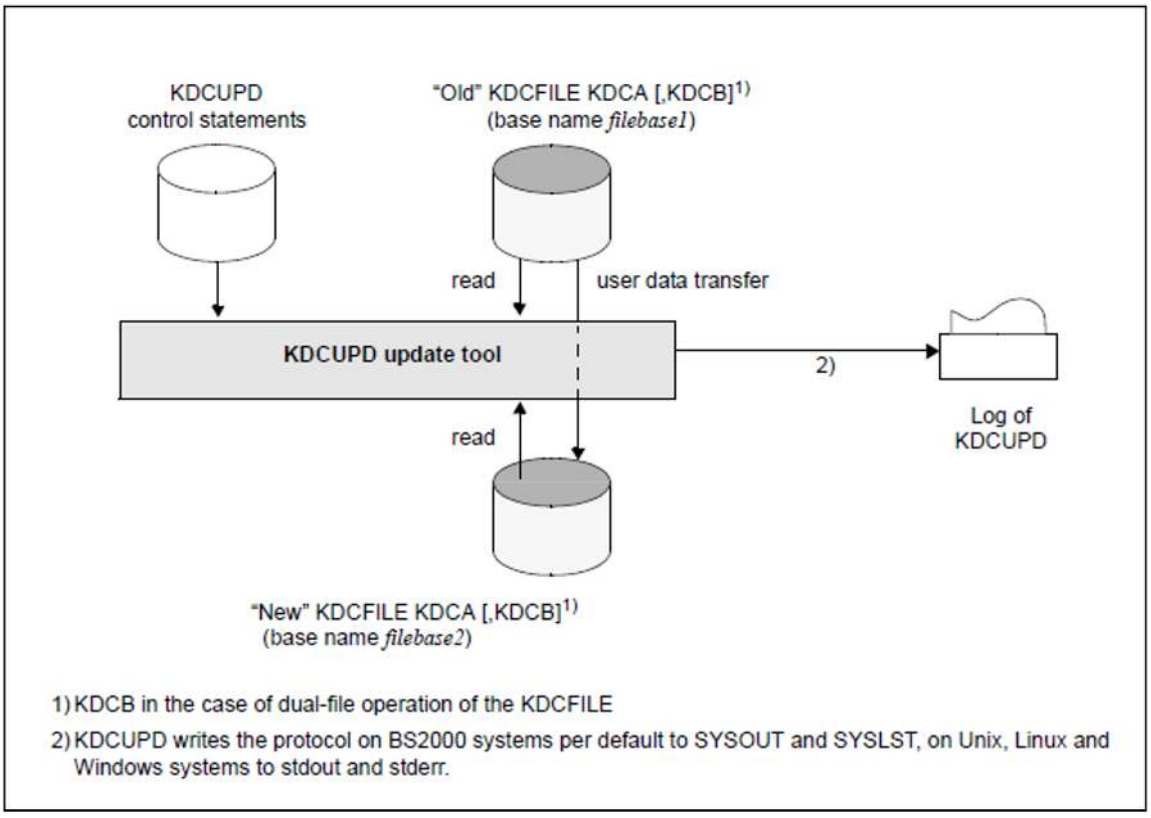


Figure 20: Updating the KDCFILE

1) Create the new KDCFILE with KDCDEF

You can change the configuration of your application in the KDCDEF run, i.e. you can define new partner applications and connections, for example, delete existing ones or change application properties, etc.

It is possible to switch from single-file to dual-file operation. In addition, the “new” KDCFILE can be divided into several files, or the number of files can be changed. When dual-file operation is used and/or the new KDCFILE is distributed across several files, all these files must exist and KDCUPD must be able to access them.

2) Terminate the application normally

Terminate the application normally, see "Prerequisite for using KDCUPD".

The application must be terminated normally before calling KDCUPD so that the old KDCFILE is in a consistent state.

Make sure that the user log file exists and is not locked when the application is terminated. openUTM stores the user log records (LPUT calls) temporarily and does not write them to the file immediately. When the application is terminated normally, an attempt is made to write these records to the current user log file. If this attempt fails, the records remain in the old KDCFILE.
KDCUPD does **not** transfer these LPUT records to the new KDCFILE. But KDCUPD indicates with a warning message (K314) that the data get lost.

3) Copy the KDCFILE and back up the user log file

After terminating the application, you must copy or rename the current KDCFILE, for instance to OLD.KDCA. Then you assign the ‘correct’ name to the new KDCFILE.

When dual-file operation is used and/or the current KDCFILE is distributed across several files, all these files must be copied or renamed using the same filebase name, and KDCUPD must be able to access them.

Back up any existing user log file, as this file is overwritten when the system is restarted after a KDCDEF or KDCUPD run.

4) Call KDCUPD

In a subsequent KDCUPD run, specify the base name of the copied KDCFILE (including the user ID on BS2000) in the control statement KDCFILE OLD= and specify the base name of the newly generated KDCFILE in the control statement KDCFILE NEW=.

BS2000 systems:

Before calling KDCUPD you must do the following:

- The library SYSLNK.UTM.070 must be set as the tasklib:

  `/SET-TASKLIB LIBRARY=$`*`userid`*`.SYSLNK.UTM.070`

  The library of the openUTM version with which the new KDCFILE was generated must be assigned.If no library or an incorrect library was assigned, the DLL outputs a corresponding message.
- Process switch 3 should be set to OFF:

  `/MODIFY-JOB-SWITCHES OFF=3`

KDCUPD is called as follows:

```
/START-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM -
             (LIBRARY=$userid.SYSLNK.UTM.070.UTIL,ELEMENT-OR-SYMBOL=KDCUPD)
:
: KDCUPD control statements
:
END
```

Alternatively, you can also call KDCUPD via the SDF command START-KDCUPD. This command is located in the SDF UTM application area. For more detailed information, see openUTM manual “Using UTM Applications on BS2000 Systems” section "Calling UTM tools".

KDCUPD reads control statements from SYSDTA. A description of the KDCUPD control statements can be found as of "Control statements for KDCUPD".

KDCUPD outputs the procedure for logging purposes to SYSLST and/or SYSOUT (see control statement LIST in section "LIST - control the runtime log").

Unix, Linux and Windows systems:

The subdirectory DUMP must exist in the base directory of the old KDCFILE (*filebase1*) as well as in the base directory of the new KDCFILE (*filebase2*) before KDCUPD is called. DUMP is used for diagnostic purposes.

You start KDCUPD on Unix and Linux systems from the shell. You must call KDCUPD in a DOS window on Windows systems. Enter the following command to do this:

Unix and Linux systems: *utmpath* `/ex/kdcupd 1>upd.out`

Windows systems: *utmpath* `\ex\kdcupd 1>upd.out.txt`

Note the following:

- You should **always** redirect the output to *stdout* to a file (`upd.out` or `upd.out.txt` in the examples above).
- You may **not** redirect *stderr* (command mode) because KDCUPD asks you to enter the control statements via *stderr* (e.g. the ‘‘ of the KDCUPD input mode is output to *stderr*).

KDCUPD reads the control statements from *stdin*. A description of the control statements for KDCUPD can be found in section "Control statements for KDCUPD".

KDCUPD outputs the procedure for logging purposes to *stdout* and/or *stderr* (see control statement LIST in section "LIST - control the runtime log").

5) Start the application

The base name (*filebase2*) of the new KDCFILE must be specified in the start parameter FILEBASE=. Modify your start procedure if you have assigned a new base name to the new KDCFILE when creating it.

If the data has been transferred to the new KDCFILE with KDCUPD and you restart the application, then every user can continue to work as if the application was terminated normally and then restarted.

8.3 Update generation for UTM cluster applications (Unix, Linux and Windows systems)

> ! Before performing the operations below, please read section "Backing up data"!

KDCUPD allows you to update and convert UTM cluster applications as follows:

- Offline update of a UTM cluster application,
  where the UTM cluster application must be shut down.
- Online update of a UTM cluster application,
  which can be performed while the UTM cluster application is running.
- Converting a UTM cluster application,
  i.e. converting a UTM cluster application from standalone to cluster and vice versa.

When performing a KDCUPD run in UTM cluster applications, you can choose between a node update and a cluster update:

- In the case of a **node update**, you update the KDCFILE of a individual node application.
- In the case of a **cluster update**, you update the UTM cluster files created during generation.

You can control these two variants using the KDCFILE and CLUSTER-FILEBASE statements.

The following sections describe the KDCUPD statements that are required for these functions and provide information on how you must first use KDCDEF to create the generation file. In addition, further activities are required depending on the situation, e.g. starting or terminating node applications or the UTM cluster application or adapting the start parameters.

For details see the relevant openUTM manual “Using UTM Applications”, chapter “UTM cluster application”.

8.3.1 Offline update of a UTM cluster application

An offline update in a UTM cluster application is necessary in the following cases:

- If you convert the UTM cluster application from a previous version to V7.0.In this case you must also regenerate the UTM cluster files (OPTION GEN=(CLUSTER, KDCFILE), see "Offline Update with cluster update".
- If you make adaptations which are listed in section "Adaptations to the UTM generation that require an offline update". For most changes, it is sufficient simply to recreate the KDCFILE (OPTION GEN=KDCFILE). However, in the case of certain adaptations it is also necessary to regenerate the UTM cluster files (OPTION GEN= (CLUSTER, KDCFILE)).

In order to perform an offline update, it is necessary to shut down all the node applications and therefore also the UTM cluster application for at least a short period.

i
- In general, the following applies: When a node application is started, the KDCFILE must not be older than the UTM cluster files.
- If you perform a conversion from a previous version to V7.0 then you must install openUTM V7.0 on all the nodes before calling KDCDEF.

Adaptations to the UTM generation that require an offline update

The table below indicates what you have to specify in the OPTION statement for the individual changes.

Type of change	KDCDEF control statements	OPTION GEN=
Switching between operation with and without users	USER	(CLUSTER, KDCFILE)
Switching between operation with and without multiple sign-on being permitted	SIGNON MULTI-SIGNON	KDCFILE
Switching between applications with and without a formatting system	FORMSYS	(CLUSTER, KDCFILE)
Changing the password history	SIGNON PW-HISTORY	KDCFILE
Changing the database systems	DATABASE, RMXA	(CLUSTER, KDCFILE)

Type of change	KDCDEF control statements	OPTION GEN=
Changing the number of LSSBs, GSSBs or ULSs	MAX LSSB, MAX GSSB, ULS	(CLUSTER, KDCFILE)
Reduction in the maximum number of services that the user is permitted to stack	MAX NRCONV	KDCFILE
Reduction in the maximum number of asynchronous services that can be opened simultaneously	MAX ASYNTASKS, second parameter	KDCFILE
Reduction in the size of the node page pool	MAX PGPOOL, first parameter value	KDCFILE
Reduction in the size of the process-specific buffer for caching restart data	MAX RECBUF, second parameter value	KDCFILE
Changing the length of the communication area	MAX KB	(CLUSTER, KDCFILE)
Reduction in the size of the standard primary working area	MAX SPAB	KDCFILE
Changing the size of the message area	MAX NB	(CLUSTER, KDCFILE)
Changing the maximum length of physical output messages	MAX TRMSGLTH	(CLUSTER, KDCFILE)
Reduction of the maximum length of the user data in LPUT records	MAX LPUTLTH	KDCFILE
Switching between UTM-S and UTM-F	MAX APPLIMODE	(CLUSTER, KDCFILE)
Increasing the number of the generated node applications	CLUSTER-NODE	(CLUSTER, KDCFILE)
Changing the names of the ULSs	ULS	(CLUSTER, KDCFILE)

Type of change	KDCDEF control statements	OPTION GEN=
Reducing the size of the cluster pagepool	CLUSTER PGPOOL, first parameter value	(CLUSTER, KDCFILE)
Changing the number of the cluster pagepool files	CLUSTER PGPOOLFS	(CLUSTER, KDCFILE)
All other changes in the CLUSTER statement except for the PGPOOL parameter	CLUSTER	(CLUSTER, KDCFILE)

Offline Update with cluster update

Proceed as follows:

1. Use the administration facilities to delete all objects that can be dynamically administered and that are no longer to be included in the new configuration.
2. Create the generation statements for a new KDCDEF run as follows: First create the statements for new objects that have been newly introduced into the application dynamically. To do this, call the online inverse KDCDEF in an active node application.
   Note that you must not create, delete or modify any more objects after you have performed the online inverse KDCDEF, otherwise the update generation is not correct.
3. Terminate the UTM cluster application.
4. Create generation statements for new objects manually or modify existing generation statements to suit your requirements.
5. Generate a new initial KDCFILE and new cluster files using the modified KDCDEF statements. When you do this, specify GEN=(CLUSTER, KDCFILE) in the KDCDEF statement OPTION, see "OPTION - manage the KDCDEF run".
6. Make the old and new UTM cluster files as well as an old KDCFILE and the new initial KDCFILE available. You may need to rename the files first.

   Use KDCUPD to perform a cluster update. This transfers user data from the UTM cluster application run to the new UTM cluster files. This data includes, for example, GSSB, ULS, the service data of users with RESTART=YES as well as the passwords of users.

   For the old and new KDCFILE, you can use either an initial KDCFILE or a KDCFILE of a node application. For this KDCUPD run, the KDCFILEs are used only for the program run and for a variety of checks. The content of the old KDCFILE is **not** transferred and the new KDCFILE is not changed.

   Perform the KDCUPD run with the following statements:

   ```
   CLUSTER-FILEBASE OLD=cluster-filebase-old,NEW=cluster-filebase-new
   KDCFILE OLD=filebase-old,NEW=filebase-new
   TRANSFER ...
   ```

   Explanation

cluster-filebase-old

Base name of the old UTM cluster files.

cluster-filebase-new

Base name of the new UTM cluster files generated by KDCDEF. KDCUPD transfers the data that is valid globally in the cluster from the old UTM cluster files to the new UTM cluster files. You use the TRANSFER statement to specify the scope of the data to be transferred.

filebase-old

Base name of an old KDCFILE in the UTM cluster application.

filebase-new

Base name of a new KDCFILE in the UTM cluster application.

7. Copy the new initial KDCFILE (see step 5) into the node-specific filebase for a node application.
8. Perform a KDCUPD run for this node application (node update) using the KDCFILE of this node as the new KDCFILE (node update).

   When you do this, specify the following KDCUPD statements:

   ```
   KDCFILE OLD=filebase-old,NEW=filebase-new
   TRANSFER ...
   ```

   During this run, transfer all the user data from the last application run of this node application into the new KDCFILE of this node application. This allows, for instance, asynchronous messages of this node application to be transferred from the old KDCFILE to the new KDCFILE.
9. Carry out steps 7 and 8 for all other node applications without delay in order to update all node applications to the same generation status.
10. Restart he UTM cluster application.

Offline Update without cluster update

If you make only such changes that are listed in the table on "Offline update of a UTM cluster application" and which have the entry KDCFILE in the column OPTION GEN= then you proceed as follows:

> Carry out steps 1 through 4 starting on "Offline update of a UTM cluster application".

> In step 5, generate a new initial KDCFILE using the modified KDCDEF statements. When you do this, specify GEN=KDCFILE in the KDCDEF statement OPTION, you must **not** specify GEN=CLUSTER!

> Carry out steps 7 through 10 on "Offline update of a UTM cluster application".

Special features in UTM-F cluster applications

In UTM cluster applications, the global UTM storage areas GSSB and ULS are also transaction-oriented in the case of UTM-F. The service data belonging to a user is saved when the user signs off.

This means that in the case of an update generation with a cluster update, it is possible to transfer the same data as in the case of a UTM-S cluster application.

> In contrast, when a node update is performed, not all the data is transferred but instead only the program versions of the load modules.

8.3.2 Online update of a UTM cluster application

When you perform an online update, only the KDCFILE is regenerated and no changes are made to the UTM cluster files. As a result, the changes to the generation apply only to the KDCFILE.

You can perform an online update while the cluster application is running, i.e. at least one node application is always active during the update.

You can make the following changes without terminating the UTM cluster application:

- Update generation of the KDCFILE for which it is not necessary to completely terminate the UTM cluster application, see below. This is the case for all changes that are not listed in the table on "Offline update of a UTM cluster application".
- Increase in the size of the cluster page pool, see "Increasing the size of the cluster page pool".
- Change to the application program, see "Change to the application program".

8.3.2.1 Update generation of the KDCFILE

An update generation of the KDCFILE for UTM cluster application will be necessary, for instance, if spare capacity for dynamic objects has been exhausted or if changes must be made to the configuration that are not possible using the dynamic administration facilities. Examples include entering additional transport system end points or partner applications for distributed processing or increasing the size of the cache, or page pool.

> **CAUTION!**
> If you modify the KDCFILE without terminating the UTM cluster application then you must not change the order of the TAC statements. Otherwise services may be terminated abnormally on service restarts. As a result, you must append new TAC statements at the end and must not delete any TAC statements.You should also not modify the RESTART parameter in the USER statements.

Proceed as follows to perform an update generation of the KDCFILE:

1. Use the administration facilities to delete all objects that can be dynamically administered and that are no longer to be included in the new configuration.
2. Create the generation statements for a new KDCDEF run as follows: First create the statements for new objects that have been newly introduced into the application dynamically. To do this, call the online inverse KDCDEF in an active node application.
   Note that you must not create, delete or modify any more objects after you have performed an online inverse KDCDEF, otherwise the update UTM generation will not be correct.
3. Create generation statements for new objects manually or modify existing generation statements to suit your requirements.

   When generating the new KDCFILE, specify the filebase name of the cluster user file currently opened by the running openUTM application under CLUSTER USER-FILEBASE=, see "CLUSTER - Define global properties of a UTM cluster application".
4. Generate a new initial KDCFILE using the modified KDCDEF statements. To do this specify OPTION GEN=KDCFILE. You must **not** specify GEN=CLUSTER!.
5. Terminate one of the node applications normally (e.g. using KDCSHUT GRACE or WinAdmin/WebAdmin).
6. Rename the KDCFILE of the terminated node application (in preparation for the KDCUPD run).
7. Copy the new initial KDCFILE created in step 4 into the node-specific filebase for the terminated node application from step 6.
8. Perform a KDCUPD run for this node application using the KDCFILE of this node as the new KDCFILE (node update).

   When you do this, specify the following KDCUPD statements:

   ```
   KDCFILE OLD=filebase-old,NEW=filebase-new
   TRANSFER ...
   ```

   Explanation

   filebase-old
   : Base name of the node application's old KDCFILE.

   filebase-new
   : Base name of the new KDCFILE generated using KDCDEF and copied for the node application.

   KDCUPD transfers the data from the old KDCFILE to the node application's new KDCFILE. You use the TRANSFER statement to specify the scope of the data to be transferred. This allows, for instance, asynchronous messages of this node application to be transferred from the old KDCFILE to the new KDCFILE.

9. Restart this node application using the new KDCFILE that has been prepared as described.

   When you restart the node application, the values of the start parameters that apply globally in the cluster are taken over from the running UTM cluster application. The sources for these are as follows:

   - the administration journal in which recent global administration actions are logged,
   - the file containing the online copy of the management data of the UTM cluster application from which older changes are taken over.

10. Carry out steps 5 through 9 for all other node applications without delay in order to update all node applications to the same generation status.

- Note that global administration of all applications of a cluster and an online inverse KDCDEF run are not possible until all active node applications have been updated to the same generation status. Local administration of individual node applications, however, can be carried out at any time.

CAUTION!
After a node application has been restarted on the basis of a newly generated KDCFILE, it is not possible to start other node applications using a KDCFILE from an older generation run.

8.3.2.2 Increasing the size of the cluster page pool

You can increase the size of the cluster page pool and/or modify the warning level for the cluster page pool while the UTM cluster application is running. To do this, you, in principle, perform an update generation of the KDCFILE as described in section "Update generation of the KDCFILE". However, you should note the following:

- Enter the new values for the size and/or warning level in the PGPOOL operand of the CLUSTER statement. You may only increase the size of the cluster page pool. It is not possible to reduce the size online!
- Perform the KDCDEF run. When you do this, enter only OPTION GEN=KDCFILE. You must **not** specify GEN=CLUSTER!
- Make sure that sufficient disk storage space is available for the enlarged cluster page pool files since this is not checked at generation time.

The remaining steps are similar to the procedure described above (see steps 5 through 9 in section "Update generation of the KDCFILE"), i.e. you update all the active node applications to the current generation state one after the other.

The change to the warning level or the increase in the size of the cluster page pool takes effect as soon as all the running node applications have been restarted with the newly generated KDCFILE.

The size of the cluster page pool files is increased by the running UTM cluster application and the additional pages are taken into account when new pages are reserved for the relevant nodes.

> If there is not enough space available in order to increase the size of the cluster page pool then the UTM cluster application aborts the increase with an error message and continues to work with the original file size. In this case, the disk storage space for individual cluster page pool files that have already been increased is not reduced again but also not used!

8.3.2.3 Change to the application program

You can add new program units to the UTM cluster application or modify existing program units without the need to terminate the entire UTM cluster application. In order to do this, you should always generate the application in such a way that the ROOT table module is dynamically loaded when the application is started. You should avoid statically linking program units.

1. To add new program units that are not yet assigned to any shared object already existing in the application, create a new ROOT table module in a KDCDEF run.
   This can be done while the application is running.
2. Then compile the ROOT table module and link the application program again as necessary.
   This can be done irrespective of whether node applications of the UTM cluster application are active or not.
3. Then close a the node applications and replace the application program.
4. Restart the node application with the new application program.
5. Repeat the steps 3 and 4 successively for all other node applications.

Please note:
If you define a new shared object, you must also generate a new initial KDCFILE, copy this to the node applications and perform a KDCUPD run, see section "Update generation of the KDCFILE".

Until this action has been completed for all node applications, the node applications of the cluster use different versions of the application program. This may affect the behavior of the application. It is for instance possible that a particular program unit is called in one node application but not in another node application.

> If the modified application program uses new programs and/or new transaction codes, then you can add these to the configuration using the dynamic administration capabilities, e.g. directly before or after the replacement of the application program.

8.3.3 Converting a UTM cluster application

KDCUPD allows you to convert UTM cluster applications when performing the following actions:

- Conversion from a standalone application V7.0 to a UTM cluster application V7.0, see below.
- Conversion from a UTM cluster application V7.0 to a standalone UTM application V7.0, see "Converting a UTM cluster application to a standalone UTM application"

8.3.3.1 Conversion from a standalone UTM application to a UTM cluster application

Standalone UTM applications can be converted directly to UTM cluster applications in the case of UTM applications as of version 7.0.

If you want to convert a standalone UTM application < V7.0 into a UTM cluster application then you must first convert it into a standalone application version 7.0.

Activities before conversion

Before conversion, you have to check the following points and make adaptations if necessary:

- Application code

  It is not necessary to adapt the code of the application unless

  - the global memory areas AREA and shared memories are used, because these lose their global nature in the UTM cluster application.
  - other application-specific resources are used whose functionality must be available across the entire cluster when migrating to a UTM cluster application.
- UPIC clients
  - You only need to adapt the UPICFILE for UPIC clients whose paths to the UTM applications have been configured statically in the UPICFILE.
  - You must adapt the UPICFILE and the client program for UPIC clients that dynamically configure their paths to the UTM applications using SET calls.

For detailed information on adapting UPIC clients, refer to the manual „openUTM-Client for the UPIC Carrier System”.

Performing the Conversion

A standalone UTM application is running on **one** node. It is to be converted to a UTM cluster application that is to run on **several** different nodes:

Proceed as follows:

1. First install openUTM V7.0 on all the nodes.
2. Extend the generation statements for a new KDCDEF run as follows:

   > Define the cluster-specific name prefix as the storage location for the files that are global to the cluster (CLUSTER statement, CLUSTER-FILEBASE operand).

   > Configure each node with one CLUSTER-NODE statement per node.
3. Run the KDCDEF utility with OPTION GEN=(CLUSTER,KDCFILE):

   The new initial KDCFILE is generated and the UTM cluster files of the UTM cluster application are created.
4. Terminate the standalone UTM application on the computer.
5. Back up the KDCFILE of the standalone application for the subsequent KDCUPD run.
6. Make the new cluster files generated by KDCDEF, the initial KDCFILE and the old KDCFILE available under the base names which you specify in CLUSTER-FILEBASE NEW=, KDCFILE OLD= and KDCFILE NEW= in the cluster update.

7. Use KDCUPD to perform a cluster update. When you do this, the data that applies globally at cluster level is transferred from the old KDCFILE of the standalone application to the UTM cluster files.

   > Make the new UTM cluster files and the old and new KDCFILE available under the base names specified below. For the new KDCFILE, you can use either the initial KDCFILE or the KDCFILE of a node application. In this case, the KDCFILEs are only used for various checks. The content of the new KDCFILE is not changed.

   > Perform the KDCUPD run with the following statements:

   ```
   CLUSTER-FILEBASE NEW=cluster-filebase
   KDCFILE OLD=filebase-old,NEW=filebase-new
   TRANSFER ...
   ```

   Explanation

   cluster-filebase

   Base name of the new UTM cluster files generated by KDCDEF. KDCUPD transfers the data that applies globally at cluster level from the old KDCFILE of the standalone application to the UTM cluster files. You use the TRANSFER statement to specify the scope of the data to be transferred.

   filebase-old

   Base name of the old KDCFILE in the standalone application.

   filebase-new

   Base name of the new KDCFILE in the UTM cluster application.

8. Copy the initial KDCFILE for each node application into the corresponding node specific filebase.
9. Perform a KDCUPD run with the following statements for a node application's KDCFILE:

   ```
   KDCFILE OLD=filebase-old,NEW=filebase-new
   TRANSFER ...
   ```

   Explanation

   filebase-old

   Base name of the old KDCFILE in the standalone application.

   filebase-new

   Base name of the node application's KDCFILE. KDCUPD transfers the data from the old KDCFILE to the node application's KDCFILE. You use the TRANSFER statement to specify the scope of the data to be transferred.

   > **! CAUTION!**
   > You can only perform a KDCUPD run for a node application!

   When you do this, the data that applies locally at node level is transferred from the old KDCFILE of the standalone application to the node application's new KDCFILE.

10. Make the UTM cluster files available at the storage location that you have specified in

    the start parameter CLUSTER-FILEBASE. Make the node KDCFILEs available at the storage location that you have specified in the KDCDEF statement CLUSTER-NODE. These storage locations must be present on a medium that can be accessed by all the nodes.

11. In all node applications, replace the start parameters

`.UTM START FILEBASE=<filebase>`

required for standalone applications by the statement

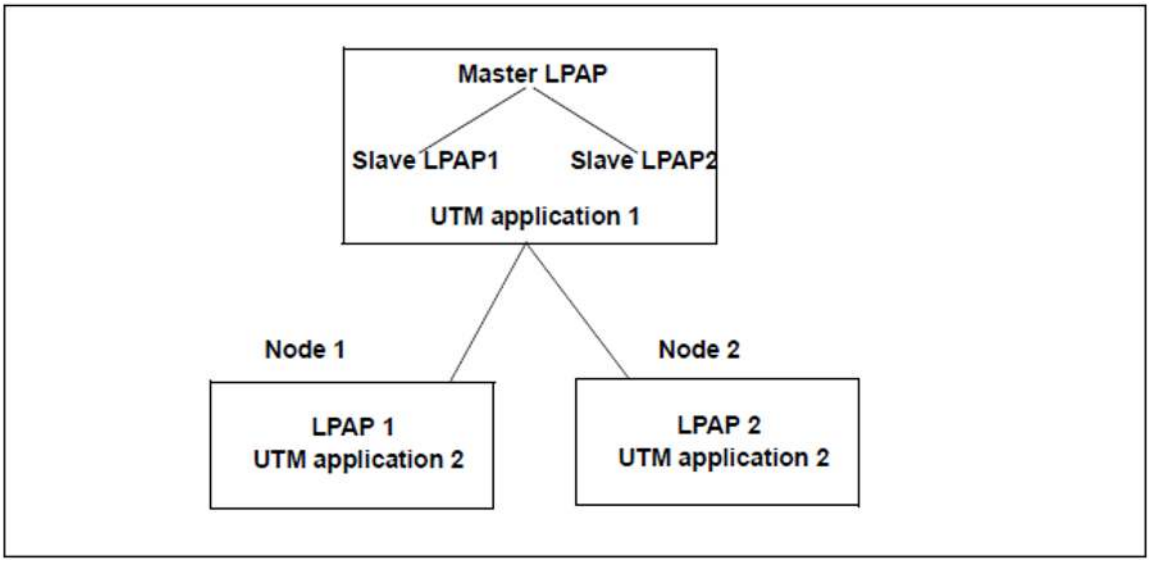
`.UTM START CLUSTER-FILEBASE=<cluster-filebase>.`

See also the section „Start parameters for openUTM“ in the corresponding openUTM manual “Using UTM Applications” for details.

12. Start the first node application with the KDCFILE modified in step 9.
13. Start the other node applications.

Adapting other UTM applications that communicate with the UTM cluster application using OSI TP or LU6.1

If UTM application 1 communicates with UTM application 2 via OSI TP or LU6.1, and you wish to convert UTM application 2 to a UTM cluster application, you should generate LPAP bundles in UTM application



The master LPAP is addressed by application 1. The master LPAP sends messages to the slave LPAPs of the connected nodes on which application 2 is running in sequence. In this event, the LPAP bundle acts as a static load distributor.

For detailed information see section "LU6.1-LPAP bundles" and section "OSI-LPAP bundles".

8.3.3.2 Converting a UTM cluster application to a standalone UTM application

If you want to convert a UTM cluster application of V7.0 into a standalone V7.0 application then you can perform either a cluster update or a node update, but not both. This is due to the fact that KDCUPD is able to transfer data to a newly generated KDCFILE **once only**.

When you perform a cluster update, you can only transfer data that applies globally to the cluster such as passwords, GSSB, ULS and service-specific data. In the case of a node update, you can only transfer local, node-level data such as TLS, asynchronous messages etc.

1. Use KDCDEF to generate the KDCFILE for the standalone application. To do this, specify OPTION ... GEN=KDCFILE. You must not specify GEN=CLUSTER.
2. Perform either a **cluster update** or a **node update**:

   Cluster update

   - Terminate the UTM cluster application.
   - Make the old UTM cluster files and the old and new KDCFILE available under the base names specified below. For the old KDCFILE, you can use either the initial KDCFILE or the KDCFILE of a node application. In this case, the KDCFILEs are only used for various checks. The content of the old KDCFILE is **not** transferred.
   - Perform the KDCUPD run with the following statements:

     ```
     CLUSTER-FILEBASE OLD=cluster-filebase-old
     KDCFILE OLD=filebase-old,NEW=filebase-new
     TRANSFER ...
     ```

     When you do this, KDCUPD transfers the global, cluster-level data such as passwords, locales (BS2000 systems), GSSB, ULS and service-specific data from the UTM cluster files to the KDCFILE of the new standalone application.

     Explanation

     cluster-filebase-old
     Base name of the old UTM cluster files. KDCUPD transfers the data that applies globally at cluster level to the KDCFILE of the new standalone application. You use the TRANSFER statement to specify the scope of the data to be transferred.

     filebase-old
     Base name of the selected KDCFILE in the UTM cluster application (initial KDCFILE or KDCFILE of a node application).

     filebase-new
     Name of the KDCFILE of the standalone application. KDCUPD transfers the data of the UTM cluster files to the new KDCFILE.

   Node update

   - Terminate all node applications except for one.
   - At the node application that is still running, perform an online import for the other node applications in order to be able to transfer as much of the node-specific data as possible.
   - Terminate this node application.

- Perform the KDCUPD run with the following statements:

  ```
  KDCFILE OLD=filebase-old,NEW=filebase-new
  TRANSFER ...
  ```

  Explanation

  filebase-old
  : Name of the KDCFILE of the lastly terminated node application.

  filebase-new
  : Name of the KDCFILE of the standalone application. KDCUPD transfers the data from the KDCFILE of the node application to the KDCFILE of the standalone application.

  You use the TRANSFER statement to specify the scope of the data to be transferred.

3. Start the standalone application with the new KDCFILE.

8.4 Update generation with transfer from 32-bit to 64-bit architecture

On 64-bit platforms, KDCUPD is also able to transfer data from a 32-bit UTM application to a 64-bit UTM application. The transfer is always performed from the 32-bit UTM application to the 64-bit application and is supported only by the 64-bit variant of the utility program KDCUPD.

When doing so, please note the following:

- KDCUPD is only able to process UTM applications on the same system platform.
- The previous 32-bit UTM application and the 64-bit UTM application to which the data is to be transferred must be of the same version.
- If the 32-bit UTM application was run with a predecessor version < V7.0 then the transfer must be performed in two stages, see following example for openUTM V6.5:

    1. Transfer from openUTM V6.5 (32-bit) to openUTM V7.0 (32-bit) using the KDCUPD V7.0 32-bit program.
    2. Transfer from openUTM V7.0 (32-bit) to openUTM V7.0 (64-bit) using the KDCUPD V7.0 64-bit program.
- In the case of UTM cluster applications, the change of architecture must be performed both for the node update and the cluster update.

Steps

The steps are the same as when performing transfers between 32-bit platforms, see for example section "Update generation for standalone UTM applications".

KDCUPD automatically detects the change of architecture when the transfer operation is performed, i.e. no special control statements are required for this operation. More specifically, the following applies:

- You enter the directory containing the previously used 32-bit KDCFILE in the OLD parameter of the TRANSFER statement.
- You enter the directory for the newly generated 64-bit KDCFILE in the NEW parameter of the TRANSFER statement.

When this change of architecture is performed, KDCUPD outputs the message K841.

> **CAUTION!**
> All user data, e.g. GSSB, LSSB, TLS, ULS content and the KB program areas, is taken over unchanged at the binary level because KDCUPD has no information about the internal structure of the user data employed by the user!

8.5 Control statements for KDCUPD

KDCUPD knows the following control statements for data transfer:

Statement	Meaning
TRANSFER	Control the transfer of the data from the old KDCFILE to the new KDCFILE
KDCFILE	Specify the base name of the newly generated and the old KDCFILE
CLUSTER-FILEBASE	Specify base name of old and new UTM cluster files *(Unix, Linux and Windows systems)*
CATID	The catid of the old and new KDCFILE are specified on BS2000 systems
LIST	Control the runtime log
END	Terminate input and start processing

KDCUPD knows the following control statements for the consistency check:

Statement	Meaning
CHECK	Check the consistency of the KDFILE of a UTM application
LIST	Control the runtime log
END	Terminate input and start processing

The control statements are read from SYSDTA on BS2000 systems or from *stdin* (command prompt) on Unix, Linux and Windows systems.The control statements KDCFILE, CATID, LIST and CHECK may only be entered once per KDCUPD run. The input must be contained in a single line. Multiple (separate single) lines can be entered for the control statement TRANSFER.

Example

1. KDCUPD is to transfer the data from the old KDCFILE (base name `BOOK01`) of a standalone UTM-S application to the new KDCFILE (base name `BOOK02`). All data is to be transferred except for the asynchronous messages intended for the communication partners (LPAP and LTERM) that are still located in the message queues of the old KDCFILE. KDCUPD is to output the successful transfer messages only to SYSLST (BS2000 systems) or *stdout* (Unix, Linux and Windows systems).

BS2000 systems	Unix, Linux and Windows systems

`*KDCFILE NEW=BUCH02,OLD=BUCH01`
`*CATID OLD=(20SN,20PN),NEW=(20SN,20PN)`
`*TRANSFER ASYNLPAP=NO`
`*TRANSFER ASYNTERM=NO`
`*LIST PROTOCOL=SYSLST`
`*END`
	`*KDCFILE NEW=BUCH02,OLD=BUCH01`
	
	`*TRANSFER ASYNLPAP=NO`
	`*TRANSFER ASYNTERM=NO`
	`*LIST PROTOCOL=STDOUT`
	`*END`

2. If you only enter the mandatory parameters in all statements, then KDCUPD attempts to transfer everything and outputs the log to SYSOUT and SYSLST or to *stdout* and *stderr*. You must specify the following to transfer all data:

BS2000 systems	Unix, Linux and Windows systems

`*KDCFILE NEW=BUCH02,OLD=BUCH01`
`*CATID OLD=(20SN,20PN),NEW=(20SN,`
`20PN)`
`*END`
	`*KDCFILE NEW=BUCH02,OLD=BUCH01`
	
	`*END`

8.5.1 CATID - define Catid of the old and the new KDCFILE

This statement only applies to BS2000 systems.

The CATID statement specifies the catalog ID of the old and new KDCFILE. You must specify at least one of the operands OLD or NEW.

`CATID`	`[ OLD=(catalog_A,catalog_B) ]` `[,NEW=(catalog_A,catalog_B) ]`

OLD= (catalog_A,catalog_B)

Catids for the old (previously used) KDCFILE

NEW= (catalog_A,catalog_B)

Catids for the new KDCFILE If you specify only *catalog_A*, this Catid is assigned to all parts of the KDCFILE. If you are working with Catids, the base name (*filebase1/2*) must be specified in the KDCFILE statement without a catalog ID.

8.5.2 CHECK - check the consistency of the KDCFILE

The CHECK statement causes KDCUPD to check the KDCFILE file(s) of an application for consistency (completeness, identical generation and processing status). No data is transfered.

`CHECK`	`filebase`

filebase Base name of the KDCFILE
(KDCDEF control statement MAX KDCFILE=*filebase)*

BS2000 systems:
With dual-file operation, *filebase* must be specified with the catalog ID of the A files.

8.5.3 CLUSTER-FILEBASE - Specify the base names of the old and new UTM cluster files

This statement only applies on Unix, Linux and Windows systems.

You use the CLUSTER-FILEBASE statement to inform KDCUPD of the base names of the old and new UTM cluster files.

CLUSTER-FILEBASE	[NEW=cluster_filebase2] [,OLD=cluster_filebase1]

NEW= cluster_filebase2

Base name of the newly generated UTM cluster files.
You must not specify this parameter when converting a UTM cluster application to a standalone application.

OLD= cluster_filebase1

Base name of the previously used UTM cluster files.You must not specify this parameter when converting a standalone application to a UTM cluster application.

The old and new UTM cluster files must have different base names. You can do this in two ways:

- For the new KDCDEF generation, enter a base name that is different from that in the old KDCDEF generation (KDCDEF statement CLUSTER; operand CLUSTER-FILEBASE).
- In the KDCDEF generation, leave the base name in CLUSTER CLUSTER-FILEBASE= unchanged. When you do this, rename the previously used UTM cluster files before the KDCUPD run so that they have a different base name.

8.5.4 END - terminate input and start processing

The END statement terminates the entry of parameters and starts processing. The KDCUPD control statements must be terminated with END.

END	

8.5.5 KDCFILE - specify the base name of the old and new KDCFILE

The KDCFILE statement passes the base name of the old and new KDCFILE to KDCUPD.

```
KDCFILE  NEW=filebase2
        ,OLD=filebase1
```

NEW= filebase2

Base name of the newly generated KDCFILE

BS2000 systems:
In the case of dual-file operation, the catalog IDS for the A and B files can be assigned with the CATID statement. In this event, you must specify the base name without the catalog ID.

OLD= filebase1

Base name of the old KDCFILE

8.5.6 LIST - control the runtime log

The LIST statement controls output of the positive and negative transfer messages as well as messages K305 and K306 for the number of pages used in the page pool.

```
LIST   [ ERRORS={ BOTH | SYSOUT[1] | SYSLST[1] | STDERR[2] | STDOUT[2] } ]
       [ ,INFO={ SHORT | LONG | NO } ]
       [ ,PROTOCOL={ BOTH | NO | SYSOUT[1] | SYSLST[1] | STDERR[2] | STDOUT[2] } ]
```

[1]SYSLST and SYSOUT are only permitted on BS2000 systems.
[2]STDERR and STDOUT are only permitted on Unix, Linux and Windows systems

ERRORS= Controls the output of negative transfer messages

- BOTH — Data is logged to SYSOUT and SYSLST (on BS2000 systems) or to *stderr* and *stdout* on Unix, Linux and Windows systems.

  Default: BOTH
- SYSOUT — Data is logged to SYSOUT.
- SYSLST — Data is logged to SYSLST.
- STDOUT — Data is logged to *stdout.*
- STDERR — Data is logged to *stderr.*

INFO= Controls the output of messages K305 and K306 for the number of pages used in the page pool of the new KDCFILE

SHORT At the end of the log for the successful transfer messages, K306 messages are used to output an overview of the number of pages used in the page pool. An overall view and a view with the number of pages used by each of the GSSB, QUEUE, LTERM, LPAP, TAC, USER, and ASYNVG object types (if they use pages in the page pool) are output.

In the page pool usage view the following are displayed for each type of object:

GSSB	Data of the GSSB storage area
QUEUE	Asynchronous messages of the temporary queue and the management information required for administration
LTERM	TLS blocks and asynchronous messages (including management information) of the LTERM
LPAP	TLS blocks and asynchronous messages (iincluding management information) of the LPAP or OSI-LPAP
TAC	Asynchronous messages (including management information) of the TAC
USER	ULS blocks, asynchronous messages (including management informationn) and dialog service information of the user, ULS blocks of the LSES or OSI-ASS.
ASYNVG	Service data (including any LSSBs that are present) of all the user's open asynchronous services

Standard: SHORT

LONG In a K305 message, the page pool usage is also output for each object of this type after the last positive transfer message as long as at least one page pool page is used.

NO Messages K305 and K306 are deactivated.

PROTOCOL= Controls the output of positive transfer messages and the messages for the page pool usage (see INFO).

BOTH Data is logged to SYSOUT and SYSLST (on BS2000 systems) or to *stderr* and *stdout* on Unix, Linux and Windows systems.

Default: BOTH

NO No positive transfer messages are output and no messages for the page pool usage.

SYSOUT Data is logged to SYSOUT.

SYSLST Data is logged to SYSLST.

STDOUT Data is logged to *stdout.*

STDERR Data is logged to *stderr.*

8.5.7 TRANSFER - control the data transfer of the user data

The TRANSFER specifies the user data KDCUPD is to transfer to the new KDCFILE.

TRANSFER	[ASYNLPAP=YES | NO] [,ASYNTACS=YES | NO] [,ASYNTERM=YES | NO] [, DB-CREDENTIALS = YES | NO] [,DIALOGS=YES | NO] [,PASS=YES | NO] [,PROG-VER=YES | NO] [,SOCKET-DIALOGS=YES | NO] [,STORAGES=YES | NO] [,UPIC-DIALOGS=YES | NO] ***Additional operand on BS2000 systems*** [,LOCALE=YES | NO]

If you omit the TRANSFER statement, KDCUPD transfers the user data as if the value YES was specified for all the TRANSFER operands.

ASYNLPAP=

YES All asynchronous messages that have not yet been output to partner applications (distributed processing via LU6.1 / OSI TP) are also transferred.

Messages from the dead letter queue with the original destination LPAP or OSI-LPAP are only transferred if the original destinations still exist in the new generation. The transfer happens regardless of whether DEAD-LETTER-Q=YES has been generated for the LPAPs or OSI-LPAPs.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

NO These messages are not transferred.

ASYNTACS=

YES All background jobs not yet processed, including time-driven jobs and all asynchronous jobs with their data, are transferred. In addition, all messages in all TAC queues are transferred.

Messages from the dead letter queue with original destination TAC or TAC queue are taken on regardless of whether the original destination still exists in the new generation or DEAD-LETTER-Q=YES was generated for TACs.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

> **CAUTION!**
> In the case of wraparound queues, the messages which were transferred first are lost and replaced by the most recently transferred messages when the queue level is reached. No warning message is issued.

NO These jobs and data are not transferred.

Neither queued messages nor open asynchronous messages are transferred.

ASYNTERM=

YES All asynchronous messages not yet output to LTERM partners are transferred, including time-driven messages.

NO These messages are not transferred.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

DB-CREDENTIALS =

YES Database password and database user name are transferred.

The following applies to UTM cluster applications:
The parameter only applies to cluster updates.

NO Database password and database user name are not transferred.

DIALOGS=

YES The data to dialog services is transferred.
If the service is open, this includes LSSBs, KB and the last dialog message.
If the service is terminated, only the last logged dialog message is transferred.

The following applies to UTM cluster applications:

- In the case of a node update, the service-specific data of connection user IDs is taken over.
- In the case of a cluster update, the service-specific data of genuine user IDs is taken over.

NO No service-specific data is transferred.

LOCALE= Additional operand on BS2000 systems

YES KDCUPD transfers the current values of the locale of each UTM user (USER) to the new KDCFILE. The values can differ from generated values, e.g. if a user has changed his/her locale using the SIGN CL call in the application run.

NO The locales of users are not transferred, i.e. the generated values apply.

PASS=

YES Passwords are transferred from the old to the new KDCFILE. This applies to all USERs for whom a password was generated in the old and in the new KDCFILE. The following is also transferred (if generated):

- the remaining validity period of the password
- the most recently used passwords, in other words the password history
- the minimum period before the password can next be changed

In the case of users for whom no password was defined in the old KDCFILE (in contrast to the new file), the new password is retained.
If no password is generated for a user in the new KDCFILE, any existing password in the old KDCFILE is not transferred.

The following applies to UTM cluster applications:
The parameter only applies to cluster updates.

NO No passwords are transferred.

PROG-VER=

YES The current version numbers of the load modules (BS2000), shared objects (Unix and Linux systems) or DLLs (Windows systems) are transferred to the new KDCFILE.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

NO The current version numbers are not transferred.

QUEUES=

YES All temporary queues and the messages they contain are transferred from the old to the new KDCFILE.

The following applies to UTM cluster applications:
The parameter only applies to node updates.

NO The temporary queues and the messages they contain are not transferred.

SOCKET-DIALOGS=

YES The data for dialog services started by socket partners is transferred. In the case of an open service, these are the LSSBs, KB and the most recent dialog message. In the case of a terminated service, it is the most recently saved dialog message.

The following applies to UTM cluster applications:

- In the case of a node update, the service-specific data of connection user IDs is taken over.
- In the case of a cluster update, the service-specific data of genuine user IDs is taken over.

NO The data is not transferred.

STORAGES=

YES All UTM secondary storage areas, i.e. GSSB, TLS and ULS are transferred.

The following applies to UTM cluster applications:

- In the case of a node update, the TLS areas are taken over.
- In the case of a cluster update, the GSSBs and the ULS areas are taken over.

NO The UTM secondary storage area are not transferred.

UPIC-DIALOGS=

YES The data for dialog services started by UPIC clients is transferred. In the case of an open service, these are the LSSBs, KB and the most recent dialog message. In the case of a terminated service, it is the most recently saved dialog message.

The following applies to UTM cluster applications:
The parameter only applies to cluster updates.

NO The data is not transferred.

8.6 KDCUPD runtime log and messages

The update tool KDCUPD creates a runtime log that contains the following important information in addition to the parameters specified:

- Specifications on the data that was transferred.
- Specifications on the data that could not be transferred (these messages are marked with *).
- Brief information on the page pool usage

KDCUPD compares the generation of the old and new KDCFILE.
As a result of these checks, KDCUPD can reject transfer of individual items of user data because they are incompatible with the generation options of the new KDCFILE.
It is also possible for KDCUPD to reject transfer completely because individual generation options of the old and new KDCFILE differ so significantly that it would not be possible to start the application with the new KDCFILE and the transferred data (see also "Changing generation parameters").

The runtime log is output to SYSOUT and SYSLST or to *stdout* and *stderr* by default. The output can be controlled using the LIST statement.

The KDCUPD messages are listed in the openUTM manual ”Messages, Debugging and Diagnostics”. The causes of error and the actions to be taken in response to the UTM message are described where necessary.

On Unix, Linux and Windows systems KDCUPD uses the NLS message catalog to output its messages.

Behavior in the event of errors

If an internal error occurs, KDCUPD creates a UTM dump (on Unix, Linux and Windows systems the dump is located in the DUMP subdirectory of the base directory).
This dump can be edited using the KDCDUMP editing tool (see the openUTM manual ”Messages, Debugging and Diagnostics”).

On BS2000 systems the process switch 3 is set if KDCUPD cannot terminate itself normally due to an error.
Process switch 3 is also set when not all of the data could be transferred to the new KDCFILE because some generation components have been removed although KDCUPD terminated itself normally.

The process switch 3 is also set if KDCUPD could not run because an error occured during checking the KDCFILEs.

Diagnostic documentation

If an error message is output in relation to the execution of KDCUPD, the following documentation should be supplied or at least saved:

- UTM dump, if one was created
  In the event of a memory bottleneck, it may be the case that no dump file can be written.For UTM applications on Unix and Linux sYstems the core dump also must be logged.
- log of KDCUPD
- KDCDEF control statements for the old and new KDCFILE
  (unless prohibited for data protection reasons)
- the old KDCFILE
- the new KDCFILE in the state before the KDCUPD run
  (alternatively KDCDEF control statements)

- in the case of cluster updates:
  the old cluster files and the new cluster files in their state before the KDCUPD run

9 Glossary

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

abnormal termination of a UTM application

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

abstract syntax (OSI)

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

acceptor (CPI-C)

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

access list

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

access point (OSI)

See *service access point.*

ACID properties

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

administration

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

administration command

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

administration journal

See *cluster administration journal*.

administration program

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

administrator

User who possesses administration authorization.

AES

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

Apache Axis

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

Apache Tomcat

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

application cold start

See *cold start*.

application context (OSI)

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

application entity (OSI)

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

application entity qualifier (OSI)

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

application entity title (OSI)

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

application information

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

application process (OSI)

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

application process title (OSI)

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

application program

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

application restart

see *warm start*

application service element (OSI)

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

application warm start

see *warm start*.

association (OSI)

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

asynchronous conversation

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

asynchronous job

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

asynchronous message

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

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

asynchronous program

Program unit started by a *background job*.

asynchronous service (KDCS)

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

audit (BS2000 systems)

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

authentication

See *system access control*.

authorization

See *data access control*.

Axis

See *Apache Axis*.

background job

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

basic format

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

basic job

Asynchronous job in a *job complex*.

browsing asynchronous messages

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

bypass mode (BS2000 systems)

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

cache

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

CCR (Commitment, Concurrency and Recovery)

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

CCS name (BS2000 systems)

See *coded character set name*.

client

Clients of a *UTM application* can be:

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

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

client side of a conversation

This term has been superseded by *initiator*.

cluster

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

cluster administration journal

The cluster administration journal consists of:

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

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

cluster configuration file

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

cluster filebase

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

cluster GSSB file

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

cluster lock file

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

cluster page pool

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

cluster start serialization file

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

cluster ULS file

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

cluster user file

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

coded character set name (BS2000 systems)

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

cold start

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

communication area (KDCS)

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

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

communication end point

see *transport system end point*

communication resource manager

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

configuration

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

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

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

confirmation job

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

connection bundle

see *LTERM bundle.*

connection user ID

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

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

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

contention loser

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

contention winner

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

conversation

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

conversation ID

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

CPI-C

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

Cross Coupled System / XCS

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

data access control

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

data space (BS2000 systems)

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

dead letter queue

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

DES

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

dialog conversation

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

dialog job, interactive job

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

dialog message

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

dialog program

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

dialog service

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

dialog step

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

dialog terminal process (Unix , Linux and Windows systems)

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

distributed processing

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

distributed transaction

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

distributed transaction processing

Distributed processing with *distributed transactions.*

dynamic configuration

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

encryption level

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

event-driven service

This term has been superseded by *event service*.

event exit

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

event function

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

event service

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

filebase

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

10 Abbreviations

Please note: Some of the abbreviations used here derive from the German acronyms used in the original German product(s).

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AES	Advanced Encryption Standard
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder - Loader - Starter (BS2000 systems)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Coded Character Set
CCSN	Coded Character Set Name
CICS	Customer Information Control System
CID	Control Identification
CMX	Communication Manager in Unix, Linux and Windows Systems
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000 systems)
DB	Database
DBH	Database Handler
DC	Data Communication
DCAM	Data Communication Access Method
DES	Data Encryption Standard

DLM	Distributed Lock Manager (BS2000 systems)
DMS	Data Management System
DNS	Domain Name Service
DP	Distribted Processing
DSS	Terminal (Datensichtstation)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans™
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GCM	Galois/Counter Mode
GSSB	Global Secondary Storage Area
HIPLEX®	Highly Integrated System Complex (BS2000 systems)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IFG	Interactive Format Generator
ILCS	Inter-Language Communication Services (BS2000 systems)
IMS	Information Management System (IBM)
IPC	Inter-Process Communication
IRV	International Reference Version
ISO	International Organization for Standardization
Java EE	Java Platform, Enterprise Edition
JCA	Java EE Connector Architecture
JDK	Java Development Kit
KAA	KDCS Application Area
KB	Communication Area

KBPRG	KB Program Area
KDCADMI	KDC Administration Interface
KDCS	Compatible Data Communication Interface
KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000 systems)
LSSB	Local Secondary Storage Area
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000 systems)
NB	Message Area
NEA	Network Architecture for BS2000 Systems
NFS	Network File System/Service
NLS	Native Language Support
OLTP	Online Transaction Processing
OML	Object Module Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Process Identification
PIN	Personal Identification Number
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Computer Center Accounting Procedure
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption algorithm according to Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000 systems)

RTS	Runtime System
SAT	Security Audit Trail (BS2000 systems)
SECOS	Security Control System
SEM	SE Manager
SGML	Standard Generalized Markup Language
SLU	Secondary Logical Unit
SM2	Software Monitor 2
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primary Working Area
SQL	Structured Query Language
SSB	Secondary Storage Area
SSL	Secure Socket Layer
SSO	Single Sign-On
TAC	Transaction Code
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-Specific Long-Term Storage
TLS	Transport Layer Security
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaction Mode)
TPR	Privileged Function State in BS2000 systems (Task Privileged)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Non-Privileged Function State in BS2000 systems (Task User)
TX	Transaction Demarcation (X/Open)

UDDI	Universal Description, Discovery and Integration
UDS	Universal Database System
UDT	Unstructured Data Transfer
ULS	User-Specific Long-Term Storage
UPIC	Universal Programming Interface for Communication
USP	UTM Socket Protocol
UTM	Universal Transaction Monitor
UTM-D	UTM Variant for Distributed Processing in BS2000 systems
UTM-F	UTM Fast Variant
UTM-S	UTM Secure Variant
UTM-XML	openUTM XML Interface
VGID	Service ID
VTSU	Virtual Terminal Support
WAN	Wide Area Network
WS4UTM	Web-Services for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (X/Open interface for acess to the resource manager)
XAP	X/OPEN ACSE/Presentation programming interface
XAP-TP	X/OPEN ACSE/Presentation programming interface Transaction Processing extension
XATMI	X/Open Application Transaction Manager Interface
XCS	Cross Coupled System
XHCS	eXtended Host Code Support
XML	eXtensible Markup Language

11 Related publications

You will find the manuals on the internet at https://bs2manuals.ts.fujitsu.com.

openUTM documentation

openUTMConcepts and Functions

User Guide

openUTM Programming Applications with KDCS for COBOL, C and C++

Core Manual

openUTM Generating Applications

User Guide

openUTM Using UTM Applications on BS2000 Systems

User Guide

openUTM Using UTM Applications on Unix, Linux and Windows Systems

User Guide

openUTM Administering Applications

User Guide

openUTM Messages, Debugging and Diagnostics on BS2000 Systems

User Guide

openUTM Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems

User Guide

openUTM Creating Applications with X/Open Interfaces

User Guide

openUTM XML for openUTM

openUTM Client (Unix systems) for the OpenCPIC Carrier System
Client-Server Communication with openUTM

User Guide

openUTM Client for the UPIC Carrier System
Client-Server Communication with openUTM

User Guide

openUTM WinAdmin
Graphical Administration Workstation for openUTM

Description and online help system

openUTM WebAdmin
Web Interface for Administering openUTM

Description and online help system

openUTM, openUTM-LU62
Distributed Transaction Processing between openUTM and CICS, IMS and LU6.2 Applications

User Guide

openUTM (BS2000)
Programming Applications with KDCS for Assembler
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for Fortran
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for Pascal-XT
Supplement to Core Manual

openUTM (BS2000)
Programming Applications with KDCS for PL/I
Supplement to Core Manual

WS4UTM (Unix systems and Windows systems**)**
WebServices for openUTM

Documentation for the openSEAS product environment

BeanConnect

User Guide

openUTM-JConnect
Connecting Java Clients to openUTM

User documentation and Java docs

WebTransactions
Concepts and Functions

WebTransactions
Template Language

WebTransactions
Web Access to openUTM Applications via UPIC

WebTransactions
Web Access to MVS Applications

WebTransactions
Web Access to OSD Applications

Documentation for the BS2000 environment

AID Advanced Interactive Debugger Core Manual

User Guide

AID Advanced Interactive Debugger Debugging of COBOL Programs

User Guide

AID Advanced Interactive Debugger Debugging of C/C++ Programs

User Guide

BCAM BCAM Volume 1/2

User Guide

BINDER
User Guide

BS2000 OSD/BC Commands Volume 1 - 7

User Guide

BS2000 OSD/BC Executive Macros

User Guide

BS2IDE
Eclipse-based Integrated Development Environment for BS2000
User Guide and Installation Guide
Web page: *https://bs2000.ts.fujitsu.com/bs2ide/*

BLSSERV
Dynamic Binder Loader / Starter in BS2000/OSD

User Guide

DCAM
COBOL Calls

User Guide

DCAM
Macros

User Guide

DCAM
Program Interfaces

Description

FHS
Format Handling System for openUTM, TIAM, DCAM

User Guide

IFG for FHS
User Guide

HIPLEX AF
High-Availability of Applications in BS2000/OSD

Product Manual

HIPLEX MSCF
BS2000 Processor Networks

User Guide

IMON
Installation Monitor

User Guide

MT9750 (MS Windows)
9750 Emulation under Windows
Product Manual

OMNIS/OMNIS-MENU
Functions and Commands

User Guide

OMNIS/OMNIS-MENU
Administration and Programming

User Guide

OSS (BS2000)
OSI Session Service
User Guide

openSM2
Software Monitor

User Guide

RSO
Remote SPOOL Output

User Guide

SECOS
Security Control System

User Guide

SECOS
Security Control System

Ready Reference

SESAM/SQL
Database Operation

User Guide

TIAM
User Guide

UDS/SQL
Database Operation

User Guide

Unicode in BS2000/OSD
Introduction

VTSU
Virtual Terminal Support

User Guide

XHCS
8-Bit Code and Unicode Support in BS2000/OSD

User Guide

Documentation for the Unix, Linux and Windows system environment

CMX V6.0 (Unix systems)
Betrieb und Administration (only available in German)
User Guide

CMX V6.0
Programming CMX Applications
Programming Guide

OSS (UNIX)
OSI Session Service
User Guide

PRIMECLUSTER™
Concepts Guide (Solaris, Linux)

openSM2

The documentation of openSM2 is provided in the form of detailed online help systems, which are delivered with the product.

Other publications

CPI-C

X/Open CAE Specification
Distributed Transaction Processing:
The CPI-C Specification, Version 2
ISBN 1 85912 135 7

Reference Model
X/Open Guide
Distributed Transaction Processing:
Reference Model, Version 2
ISBN 1 85912 019 9

REST
Architectural Styles and the Design of Network-based Software Architectures
Dissertation Roy Fielding

TX
X/Open CAE Specification
Distributed Transaction Processing:
The TX (Transaction Demarcation) Specification
ISBN 1 85912 094 6

XATMI
X/Open CAE Secification
Distributed Transaction Processing

The XATMI Specification
ISBN 1 85912 130 6

XML
W3C specification (www consortium)

Web page: *http://www.w3org/XML*