

English



Fujitsu Software BS2000

openUTM V7.0

Using UTM Applications on Unix, Linux and Windows Systems

User Guide

Valid for:
openUTM V7.0A00

Edition November 2019

Comments... Suggestions... Corrections...

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

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

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Copyright © 2025 Fujitsu

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

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

Table of Contents

- Using UTM Applications on Unix, Linux and Windows Systems** 10
- 1 Preface** 11
 - 1.1 Summary of contents and target group** 13
 - 1.2 Summary of contents of the openUTM documentation** 14
 - 1.2.1 openUTM documentation 15
 - 1.2.2 Documentation for the openSEAS product environment 18
 - 1.2.3 Readme files 19
 - 1.3 Changes in openUTM V7.0** 20
 - 1.3.1 New server functions 21
 - 1.3.2 Discontinued server functions 25
 - 1.3.3 New client functions 26
 - 1.3.4 New functions for openUTM WinAdmin 27
 - 1.3.5 New functions for openUTM WebAdmin 28
 - 1.4 Notational conventions** 29
- 2 Creating the application program** 31
 - 2.1 Linking a UTM process on Unix and Linux systems** 33
 - 2.1.1 COBOL program units (Unix and Linux systems) 34
 - 2.1.2 Required UTM system libraries and UTM objects (Unix and Linux systems) . 36
 - 2.1.3 Shared objects (Unix and Linux systems) 38
 - 2.1.4 Calling the linkage editor (Unix and Linux systems) 39
 - 2.1.5 Linking with a makefile (Unix and Linux systems) 40
 - 2.2 Creating application programs on Windows systems** 41
 - 2.2.1 Application programs in C and C++ (Windows systems) 42
 - 2.2.1.1 Setting the options of the Visual Studio (Windows systems) 43
 - 2.2.1.2 Creating projects (Windows systems) 45
 - 2.2.1.3 Writing source programs (Windows systems) 46
 - 2.2.1.4 Compiling and linking the application (Windows systems) 47
 - 2.2.2 Creating application programs as DLLs (Windows systems) 51
 - 2.2.3 COBOL application programs on Windows systems 52
 - 2.2.4 Installing an application as a service (Windows systems) 54
- 3 Necessary files and global system resources** 58
 - 3.1 System files stderr and stdout** 59
 - 3.2 System log file SYSLOG** 60
 - 3.2.1 SYSLOG as a simple file 61
 - 3.2.2 SYSLOG as a file generation group 62
 - 3.2.3 The KDCSLOG tool for creating the SYSLOG-FGG 63
 - 3.2.3.1 Automatic size monitoring 65

3.2.4	Protection against oversized SYSLOG file	66
3.2.5	Behavior in the event of write errors	67
3.3	User log file	68
3.3.1	Response to write errors	71
3.4	DUMP directory	72
3.5	Global system resources of an application	73
3.5.1	System resources required by a UTM application	74
3.5.2	Improving performance: Changing the size of the data area in the IPC shared memory	77
4	Starting a UTM application	79
4.1	Starting a UTM application on Unix and Linux systems	80
4.2	Starting a UTM application on Windows systems	81
4.2.1	Starting with utmmain (Windows systems)	82
4.2.2	Starting as a service (Windows systems)	84
4.3	Starting a UTM application with OpenSSL	85
4.4	Starting a UTM application with TLS connections	86
4.5	Start parameter file of the application	88
4.5.1	Start parameters for openUTM	89
4.6	Cold start and warm start	100
4.7	Error messages at the application start	101
5	Terminating a UTM application	102
5.1	Terminating a UTM application normally via administration	103
5.2	The KDCSHUT tool - terminating a UTM application normally at shell level	104
5.3	Terminating a service on Windows systems	105
5.4	Terminating a UTM application abnormally	106
5.5	The KDCREM tool	108
6	UTM database application	109
6.1	Generating a UTM database connection	110
6.2	Linking a UTM database application on Unix and Linux systems	111
6.3	Linking a UTM database application on Windows systems	112
6.4	Starting and stopping a UTM database application	113
6.4.1	Start parameters for a UTM database application	114
6.4.1.1	Openstring and Closestring	115
6.4.1.2	Several instances	116
6.4.1.3	Example of Oracle start parameters	118
6.4.2	Start parameters for failover with Oracle® Real Application Clusters	120
6.4.2.1	Special issues when connecting to Oracle®	123
6.4.3	Debug parameters	125
6.4.4	Normal termination of a UTM database application	126
6.4.5	Abnormal termination of a UTM database application	127
6.5	Operating a UTM database application	128

6.5.1 User sign-on and sign-off	129
6.5.2 Diagnostics	130
7 UTM cluster application	131
7.1 Properties of a UTM cluster application	132
7.2 Installing and preparing a UTM cluster application for use	133
7.2.1 Installation	134
7.2.1.1 Installing the UTM runtime components for Unix and Linux systems	135
7.2.1.2 Installing further runtime components for Unix and Linux systems	136
7.2.2 UTM generation	137
7.2.2.1 Special UTM generation statements for UTM cluster applications	138
7.2.2.2 Generating reserve nodes	139
7.2.3 Using global memory areas	140
7.2.4 Service restart	141
7.2.5 Runtime environment	142
7.2.5.1 Files	143
7.2.5.2 Location of the files	147
7.2.6 Preparation for use	148
7.2.7 Example for Unix and Linux systems	149
7.3 Configuration of a UTM cluster application with a database	152
7.4 Starting a UTM cluster application	153
7.5 Monitoring of node applications and failure detection	155
7.5.1 Application monitoring of the node applications	156
7.5.2 Actions performed by the node applications if a failure is detected	157
7.5.3 Application data after abnormal termination of a node application	159
7.5.4 Measures taken when a node application has been terminated abnormally	160
7.5.4.1 Measures taken for users	161
7.5.4.2 Measures to be taken by the administrator	162
7.5.4.3 Node recovery	163
7.6 Online import of application data	165
7.7 Administering a UTM cluster application	166
7.7.1 Actions global to the cluster and actions local to a node	167
7.7.2 Administration journal	168
7.7.3 Reducing the number of nodes	169
7.8 Shutting down a UTM cluster application	170
7.9 Use of openUTM revision levels in the UTM cluster application	171
7.10 Diagnostics in a UTM cluster application	173
8 Working with a UTM application	174
8.1 Sign-on process with user IDs	175
8.1.1 Standard sign-on process for terminals	176
8.1.1.1 Starting the dialog terminal process by the user	177

8.1.1.2	Starting the dialog terminal process through Unix and Linux system	179
8.1.1.3	Standard sign-on dialog	180
8.1.1.4	Automatic KDCSIGN	185
8.1.2	Sign-on process for UPIC clients and TS applications	186
8.1.3	Sign-on process for OSI TP partners	188
8.1.4	Sign-on process for HTTP clients	189
8.1.5	Sign-on process in the Internet via WebServices (WS4UTM)	190
8.1.6	Sign-on process in the Internet via WebTransactions	191
8.1.7	Multiple sign-ons under one user ID	192
8.1.8	Sign-on process with sign-on services	193
8.1.8.1	Sign-on service for terminals	194
8.1.8.2	Sign-on service for TS applications	195
8.1.8.3	Sign-on service for UPIC clients	196
8.1.8.4	Possible applications for the sign-on service	197
8.1.8.5	Properties of sign-on services	198
8.1.9	Behavior in the event of locked clients/LTERM partners	199
8.2	Sign-on process without user IDs	200
8.3	Calling UTM services	201
8.3.1	Starting services from the terminal	202
8.3.2	Starting services from the UPIC client and OSI TP partner	204
8.3.3	Starting services from the HTTP client	205
8.3.4	Starting services from TS applications	206
8.3.5	Service restart	207
8.4	Authorization concept of openUTM	208
8.5	Signing off from a UTM application	210
8.6	UTM user commands for terminals	212
8.6.1	KDCOUT - output asynchronous messages	213
8.6.2	KDCDISP - output the last dialog message	214
8.6.3	KDCLAST - repeat the last output	215
8.6.4	KDCOFF - sign off from a UTM application	216
9	Exchanging programs during operation	217
9.1	Exchanging an application	218
9.1.1	Requirements exchanging an application	219
9.1.2	File generation group PROG	221
9.1.3	Process of exchanging an application	223
9.1.4	The KDCPROG tool	224
9.1.4.1	CREATE - create a file generation group (FGG)	225
9.1.4.2	INFO - query the current state of the (FGG)	226
9.1.4.3	TRANSFER - transfer utmwork to the FGG	227
9.1.4.4	SWITCH - switch the base of the file FGG	229
9.1.5	Example of exchanging an application	230

- 9.2 Exchanging shared objects** **236**
- 9.2.1 Providing and generating shared objects 237
- 9.2.2 Start of the application 238
- 9.2.3 The exchange process 239
 - 9.2.3.1 Exchanging shared objects with LOAD-MODE=STARTUP 240
 - 9.2.3.2 Exchanging shared objects with LOAD-MODE=ONCALL 241
- 9.2.4 Examples of exchanging shared objects 242
- 9.2.5 Exchanging an application with shared objects 244
- 9.2.6 Adding programs dynamically 245
- 10 Fault tolerance of openUTM** **246**
- 10.1 Errors detected by openUTM** **247**
- 10.2 Signal handling in openUTM** **248**
- 10.3 Termination of application by system crash / shutdown** **250**
- 11 Accounting** **251**
- 11.1 Definition of terms** **252**
- 11.2 Accounting phases** **255**
- 11.2.1 Calculation phase 256
- 11.2.2 Determining the variant of the accounting procedure 257
- 11.2.3 Accounting phase 259
- 11.2.4 Evaluation 261
- 11.2.5 Error situations 262
- 11.3 Accounting with distributed processing** **263**
- 11.4 Restrictions** **264**
- 12 Checking performance with openSM2 and KDCMON** **265**
- 12.1 Monitoring with openSM2** **266**
- 12.2 UTM event monitor KDCMON** **267**
- 12.2.1 Starting and stopping data entry 268
- 12.2.2 Evaluating data with KDCEVAL 269
- 12.2.3 Processing evaluation data on the PC 272
- 12.2.4 Evaluation lists 273
 - 12.2.4.1 TASKS: UTILIZATION OF THE UTM TASKS 275
 - 12.2.4.2 SUMM: TRANSACTION EVALUATION 276
 - 12.2.4.3 TIMES: DISTRIBUTION OF PROCESSING TIMES 277
 - 12.2.4.4 KCOP: UTM CALLS STATISTIC 278
 - 12.2.4.5 WAIT: WAITING TIMES 280
 - 12.2.4.6 TCLASS: EVALUATION OF THE TAC CLASSES 281
 - 12.2.4.7 TACCL: TAC SPECIFIC TAC CLASS EVALUATION 283
 - 12.2.4.8 TACPT: TAC SPECIFIC DISTRIBUTION OF PROCESSING TIMES ... 284
 - 12.2.4.9 TACLIST: TAC SPECIFIC STATISTICS 285
 - 12.2.4.10 TRACE: TASK SPECIFIC TRACES 286
 - 12.2.4.11 TRACE2: TASK PERFORMANCE TRACE 289

- 13 Load simulation with Workload Capture and Replay 292**
 - 13.1 Recording the UPIC conversation (UPIC Capture) 294**
 - 13.2 Merging trace entries 295**
 - 13.3 Preparing data using the program UpicAnalyzer 296**
 - 13.4 Replaying the UPIC session using the program UpicReplay 297**
 - 13.4.1 Adapting the UPIC configuration and UTM generation 298
 - 13.4.2 Calling UpicReplay 299
 - 13.4.3 Functioning of UpicReplay 300
- 14 Appendix 302**
 - 14.1 Installing openUTM on Unix and Linux systems 303**
 - 14.1.1 Installing UTM system functions on Unix and Linux systems 304
 - 14.1.2 Using different socket network processes (Unix and Linux systems) 305
 - 14.1.3 Installing an openSM2 connection (Unix and Linux systems) 306
 - 14.2 Installing openUTM on Windows systems 307**
 - 14.2.1 Installation of openUTM-Server (Windows systems) 308
 - 14.2.2 User environment (Windows systems) 309
 - 14.3 Structure of the openUTM installation directory 310**
 - 14.4 Environment variables of a UTM application 312**
 - 14.4.1 General environment variables for openUTM 313
 - 14.4.2 Environment variables for work processes 317
 - 14.4.3 Environment variables for using openSSL 318
 - 14.4.4 Environment variables for the KDCDUMP tool 320
 - 14.4.5 Environment variables for the X/Open interface XATMI 321
 - 14.4.6 Additional environment variables for openUTM on Unix and Linux systems 322
 - 14.4.7 Additional environment variables for openUTM on Windows systems 323
 - 14.5 Structure of the accounting records of openUTM 325**
 - 14.5.1 Structure of an accounting record 326
 - 14.5.2 Structure of a calculation record 327
 - 14.6 Processing print output without printer control (Unix and Linux systems) ... 329**
 - 14.7 Sample programs and sample applications 330**
 - 14.7.1 Sample programs for a publish / subscribe server 331
 - 14.7.2 Sample program for moving messages from the dead letter queue selectively . 332
 - 14.7.3 Sample programs for HTTP clients 333
 - 14.7.4 CPI-C sample programs 334
 - 14.7.5 Sample procedures on Unix and Linux systems 335
 - 14.7.6 Sample procedures on Windows systems 336
 - 14.7.7 Sample application on Unix and Linux systems 337
 - 14.7.8 openUTM Quick Start Kit on Windows systems 338
- 15 Glossary 339**

16 Abbreviations **372**
17 Related publications **377**

Using UTM Applications on Unix, Linux and Windows Systems

1 Preface

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

Thus, essential requirements for a modern IT infrastructure consist of, among others

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

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

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

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

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

- consistency of data and processing
- high availability of the applications
- high throughput even when there are large numbers of users (i.e. highly scalable)
- flexibility as regards changes to and adaptation of the IT system

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

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

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



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

i Wherever the term Linux system or Linux platform is used in the following, then this should be understood to mean a Linux distribution such as SUSE or Red Hat.

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

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

1.1 Summary of contents and target group

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

It contains all the information you will need to create a UTM application program in Unix , Linux and Windows systems and implement a UTM application.

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

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

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

Knowledge of the operating system is a prerequisite.

1.2 Summary of contents of the openUTM documentation

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

1.2.1 openUTM documentation

The openUTM documentation consists of manuals, the online help for the graphical administration workstation openUTM WinAdmin and the graphical administration tool WebAdmin as well as release notes.

There are manuals and release notes that are valid for all platforms, as well as manuals and release notes that are valid for BS2000 systems and for Unix, Linux and Windows systems.

All the manuals are available on the internet at <https://bs2manuals.ts.fujitsu.com>. For the BS2000 platform, you will also find the manuals on the Softbook DVD.

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

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

Introduction and overview

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

Programming

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

Configuration

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

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

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

Linking, starting and using UTM applications

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

Administering applications and changing configurations dynamically

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

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

Testing and diagnosing errors

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

Creating openUTM clients

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

- The **openUTM-Client for the UPIC Carrier System** describes the creation and operation of client applications based on UPIC. It indicates what needs to be taken into account when programming a CPI-C application and what restrictions apply compared with the X/Open CPI-C interface.

-
- The **openUTM-Client for the OpenCPIC Carrier System** manual describes how to install and configure OpenCPIC and configure an OpenCPIC application. It indicates what needs to be taken into account when programming a CPI-C application and what restrictions apply compared with the X/Open CPI-C interface.
 - The documentation for the product **openUTM-JConnect** shipped with **BeanConnect** consists of the manual and a Java documentation with a description of the Java classes.
 - The **BizXML2Cobol** manual describes how you can extend existing COBOL programs of a UTM application in such a way that they can be used as an XML-based standard Web service. How to work with the graphical user interface is described in the **online help system**.
 - You can also use the software product WS4UTM (WebServices for openUTM) to provide services of UTM applications as Web services. To do this, you need the **Web Services for openUTM** manual. Working with the graphical user interface is described in the corresponding **online help system**.

Communicating with the IBM world

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

PCMX documentation

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

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

1.2.2 Documentation for the openSEAS product environment

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

Integrating Java EE application servers and UTM applications

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

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

Connecting to the web and application integration

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

1.2.3 Readme files

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

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

Information on BS2000 systems

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

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

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor.

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

Additional product information

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

1.3 Changes in openUTM V7.0

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

1.3.1 New server functions

UTM as HTTP-Server

A UTM application can also act as an HTTP server.

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

The following interfaces have been changed:

- Generation

All systems:

- KDCDEF statement BCAMAPPL:

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

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

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

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

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

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

- New operand USER-AUTH = *NONE | *BASIC. Herewith you can specify which authentication mechanism HTTP clients must use for this access point.

- KDCDEF statement HTTP-DESCRIPTOR:

- This statement defines a mapping of the path received in an HTTP request to a TAC and additional processing parameters can be specified.

BS2000 systems:

- KDCDEF statement CHAR-SET:

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

- Programming

- KDCS communication area (KB):

- In the header of the KDCS communication area, there are new indicators for the client protocols HTTP, USP-SECURE, and HTTPS in the *kccp/KCCP* field.

- KDCS call INIT PU:

- The version of the interface has been increased to 7.
 - To obtain the complete available information, the value 372 must be specified in the KCLI field.
 - New fields for requesting (KCHTTP/http_info) and returning (KCHTTPINF/httpInfo) HTTP-specific information.

- Administration interface KDCADMI

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

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

- UTM-HTTP program interface

In addition to the KDCS interface, UTM provides an interface for reading and writing HTTP protocol information and handling the HTTP message body.

The functions of the interface are briefly listed below:

- Function *kcHttpGetHeaderByIndex()*
This function returns the name and value of the HTTP header field for the specified index.
- Function *kcHttpGetHeaderByName()*
The function returns the value of the HTTP header field specified by the name.
- Function *kcHttpGetHeaderCount()*
This function returns the number of header fields contained in the HTTP request, that can be read by the program unit.
- Function *kcHttpGetMethod()*
This function returns the HTTP method of the HTTP request.
- Function *kcHttpGetMputMsg()*
This function returns the MPUT message generated by the program unit.
- Function *kcHttpGetPath()*
This function returns the HTTP path of the HTTP request normalized with *KC_HTTP_NORM_UNRESERVED*.
- Function *kcHttpGetQuery()*
This function returns the HTTP query of the HTTP request normalized 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 *utmnetsl* is available for secure access with TLS.
If BCAMAPPL is generated with T-PROT=(SOCKET,...,SECURE) for a UTM application, a number of *utmnetsl* 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 *utmnetsl* 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 KCADMI

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 utility**

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 openUTM 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
<i>lowercase letters in italics</i>	In running text, variables and the names of data structures and fields are indicated by lowercase letters in italics.	<i>utm-installationpath</i> 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,...keyn)
<u>Underscoring</u>	Underscoring denotes the default value.	CONNECT= { YES <u>NO</u> }
abbreviated form	The standard abbreviated form of statements, operands and operand values is emphasized in boldface type. The abbreviated form can be entered in place of the full designation.	TRANSPORT-SELECTOR =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 Creating the application program

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

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

- compile the ROOT table source generated by KDCDEF
- link ROOT tables, UTM main routine, UTM system modules for the main routine KDCROOT, the C runtime system and any other runtime systems, the message module, user libraries and program units.

The program units can also be linked as shared objects. Shared objects can be exchanged dynamically during operation.



Information on exchanging programs with shared objects can be found in [chapter “Exchanging programs during operation”](#).

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

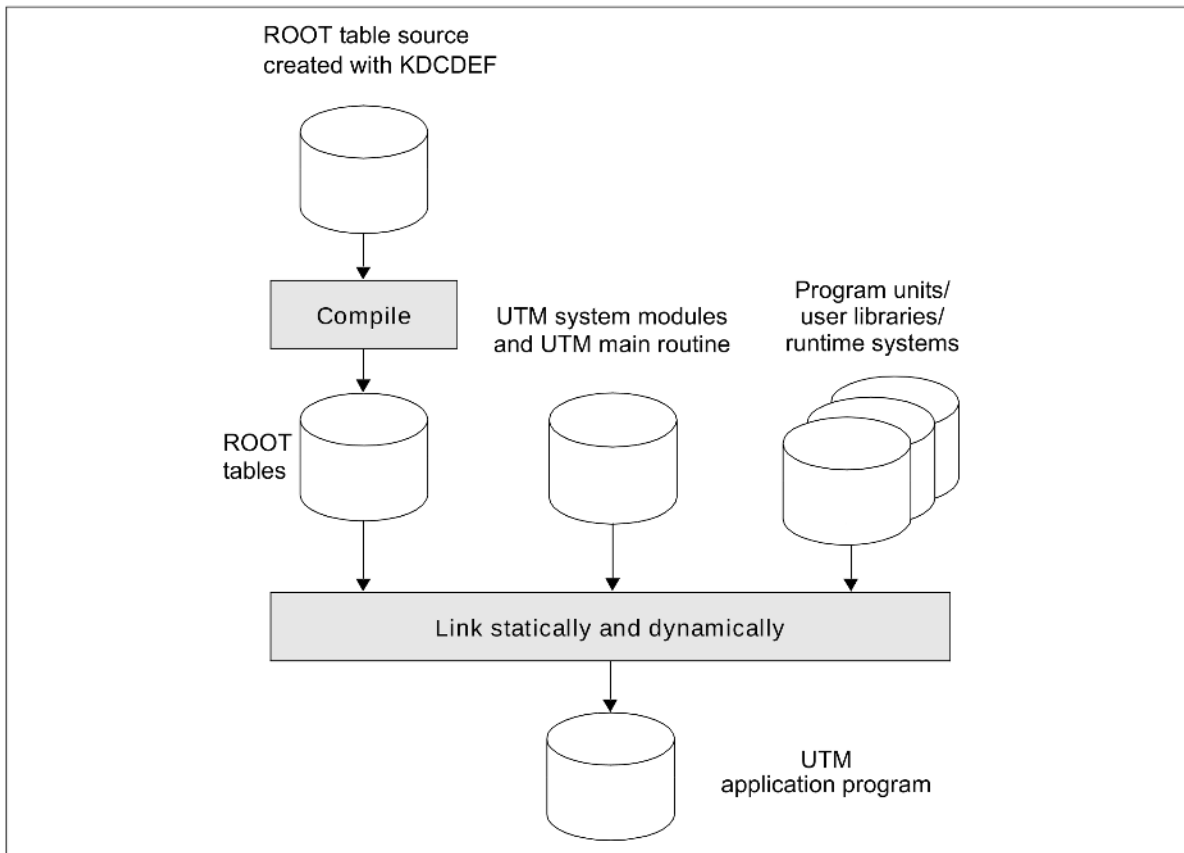


Figure 1: Overview: creating the UTM application program

Main routine KDCROOT

Based on the ROOT tables, the UTM main routine that is created during installation of openUTM and the UTM system functions, the main routine KDCROOT is generated as part of the application program during the link procedure. When running the application, KDCROOT acts as the main control program. Its tasks include:

-
- linking program units and UTM system functions
 - coordinating the execution of program units in different programming languages
 - connecting to databases

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

KDCDEF generates the ROOT table module as a C/C++ source, which you then compile using the C/C++ compiler and link to your program units, the UTM system modules, and any other modules to form an executable program, see below.

2.1 Linking a UTM process on Unix and Linux systems

For a UTM application, the UTM system libraries and UTM objects listed below must be linked in the specified order to form a work process, so that all external references can be resolved. The linked application program (work process) must be stored in the directory *filebase* under the name *utmwork*.

You can find the required UTM system libraries and system objects on your computer under *utmpath/sys*. On the AIX platform, the openUTM system libraries are provided as static libraries and on all other platforms as shared objects.

If you want to use the “program exchange” function, you have two options:

- If you want to exchange parts of the application program, you must store these parts in a shared object.
- If only the complete work process is to be exchangeable, you need do nothing. Following the linking process, you must transfer *utmwork* to the *filebase/PROG* directory using the KDCPROG tool.

Details on compiling program units, generating shared objects and linking the application program can be found in the documentation for the compiler and/or runtime system you are using.



Sample procedures for compiling and linking can be found in the supplied sample application. You can simplify this task by using the supplied sample application to create a makefile which you can use as a basis for your link procedure. See [“Linking with a makefile \(Unix and Linux systems\)”](#).

2.1.1 COBOL program units (Unix and Linux systems)

You can create COBOL programs with the Micro Focus compilers or with the Fujitsu NetCOBOL compiler.

Please note the COBOL-specific programming notes in the openUTM manual „Programming Applications with KDCS” (chapter "Additional Information for Cobol", section "Platform-specific features on Unix and Linux systems").

Environment variables for COBOL programs

If you use COBOL programs then you must set compiler-specific environment variables.

Micro Focus COBOL

Perform the following steps if you use COBOL program units with Micro Focus COBOL:

- > Call the script `<coboldir>/bin/cobsetenv`.
This script sets the required environment variables for the compiler.
- > Extend the COBCPY environment variable by adding `$UTMPATH/copy-cobol85`.
- > If you create programs based on CPIC, TX or XATMI under openUTM, extend the COBCPY environment variable as follows:
`$UTMPATH/<interface>/copy-cobol85`, where `<interface>` stands for `cpic`, `tx` or `xatmi`.
- > If you create client programs based on UPIC-L, extend the COBCPY environment variable by adding `$UTMPATH/upicl/copy-cobol85`.
- > Set the COBMODE environment variable to 64, to generate 64-bit objects.

NetCOBOL

Perform the following steps if you use NetCOBOL program units:

- > Call the script `<COBOLDIR>/config/cobol.sh`.
This script sets the required environment variables.
- > Extend the COBCOPY environment variable by adding `$UTMPATH/netcobol`.
- > Set the COB_LIBSUFFIX environment variable to `None,CPY,copy`.
- > If you create programs based on CPIC, TX or XATMI under openUTM, extend the COBCOPY environment variable as follows:
`$UTMPATH/<interface>/netcobol`, where `<interface>` stands for `cpic`, `tx` or `xatmi`.
- > If you create client programs based on UPIC-L, extend the COBCOPY environment variable by adding `$UTMPATH/upicl/netcobol`.

Note

If you use COBOL program units, you should note the following:

- There are certain compiler-specific characteristics relating to the keywords. You will find details in openUTM manual "Programming Applications with KDCS" (chapter "Additional Information for COBOL", references to "keywords").
- If you want to generate shared objects, please read [section "Shared objects \(Unix and Linux systems\)"](#).

2.1.2 Required UTM system libraries and UTM objects (Unix and Linux systems)

The libraries and objects listed below can be specified when linking an application. The configuration of the respective UTM application determines which of these components are required for execution (i.e. must also be incorporated).

- The main routine of openUTM

Here you must incorporate either the object `mainutm.o`, if only C and COBOL programs are used, or the object `mainutmCC.o`, if at least one C++ program is used.

On most systems, the application must not contain C++ and COBOL programs at the same time.

The objects `mainutm.o` and `mainutmCC.o` are contained in `utmpath/sys`. The object `mainutmCC.o` is created during the installation of openUTM if the C++ compiler is already installed.

If the C++ compiler is not installed until after openUTM or if openUTM could not find the C++ compiler during installation for some other reason, you can create `mainutmCC.o` yourself using the UTM shell script `CCmainutm`. In this case, you must enter the following command:

```
UTMPATH=utmpath \  
utmpath/shsc/CCmainutm
```

For more information on `utmpath` see "[Installing UTM system functions on Unix and Linux systems](#)".

- The main routine KDCROOT (`rootname.o`)

This module is required in every application and is created from the ROOT source program. You generate the ROOT source program (`filebase/rootname.c`) in the KDCDEF run by specifying `GEN=ROOTSRC` or `GEN=ALL` in the OPTION statement. You define `rootname` in the ROOT statement and `filebase` in the MAX statement using the `KDCFILE=` parameter.

You create KDCROOT as follows:

- Compile the ROOT source program created with KDCDEF using the C compiler. You must specify the `-I$UTMPATH/include` option here for the openUTM system includes.

If you use COBOL program units, you must also specify the `-I$COBDIR/include` option. `$COBDIR` is the installation path of the COBOL compiler.

- Store the output in `filebase/rootname.o`.

You must specify the object module `filebase/rootname.o` **before** the application-specific modules when linking the application.

- Your compiled program units

Your compiled program units can be integrated either as individual objects or as parts of object libraries. These libraries may be static or dynamic (known as shared objects).

- The UTM system libraries including administration program and code conversion tables

The library `libwork` is needed in every application program. In addition, the following applies:

- On all Unix/Linux platforms except AIX, the library `libutmconvt` is automatically loaded dynamically when starting the application. To do this, `LD_LIBRARY_PATH` or `LD_LIBRARY_PATH_64` must be set to `utmpath/sys`.
- On the AIX platform, the library `libutmconvt.so` must be specified when linking `utmwork`.

-
- OSI TP modules (optional)

If communication is to take place via OSI TP, the library `libxaptp` must be incorporated.

In addition, the OSS libraries `libossutm` and `liboss` are required. Please note that `libossutm` must always be specified **before** `liboss`, as otherwise the following error will occur when the application starts:

```
P001 Error on OSS call (o_create() .): - 1, 300, 199, 0
K060 Application run aborted; reason = XINI06.
```

- The runtime systems for C and possibly COBOL

The runtime system for the programming language C is always required. The COBOL runtime system is only needed if you have created a program unit in COBOL.

- A user-specific message module (optional)

If you have created your own message module (see the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”), you must also incorporate this object module.

- The runtime system for database system (optional)

See [section “Linking a UTM database application on Unix and Linux systems”](#) for a description of which libraries and modules you must incorporate for a UTM database link.

- Additional libraries for XML (optional)

If you want to use XML functions in program units, you will need the appropriate XML library.

For more details, see the documentation on the XML interface.

- Additional libraries for X/Open (optional)

If you have created program units using the X/Open interfaces, you will require the UTM system library `libxopen`.

In addition, the `-lm` option must be specified when linking.

- You must specify the `-lcrypt` option for Linux.

- You must always specify the `-ldl` option.

- If your application is intended to run in a 64-bit environment, ensure that the compiler generates 64-bit objects when you compile your program units and the KDCROOT module. You do this by setting the appropriate switches on the compiler (see the documentation for the compiler concerned).

2.1.3 Shared objects (Unix and Linux systems)

You must observe the following guidelines when linking *utmwork* to shared objects:

COBOL program units

If COBOL program units are dynamically loaded later as shared objects, then the COBOL run-time system must also be linked as a shared object in the program unit. Otherwise, the program unit will not be able to find the entries of the COBOL runtime system.

Micro Focus COBOL

You generate shared objects with the following call.

```
cob -z -o shared-object Cobol-objects
```

You must strictly observe the sequence of the options and specified objects.

NetCOBOL

You generate shared objects with the following call.

```
cobol -shared -dy -o shared-object Cobol-objects
```

You must strictly observe the sequence of the options and specified objects.

AIX platform

The use of shared objects by openUTM is not supported on the AIX platform.

2.1.4 Calling the linkage editor (Unix and Linux systems)

Depending on the language used to create the program units, the following linkage editors must be used:

- the C linkage editor `cc` if only C programs are contained
- the COBOL linkage editor from Micro Focus or NetCOBOL if at least one COBOL program is contained.
 - The call of the Micro Focus linkage editor is then as follows:
`cob -o utmwork shared-object UTM-systemlibraries`
 - Linking with NetCOBOL is performed using the `cobol` command. To do this, it is necessary to incorporate the following COBOL libraries:
 - `/opt/FJSVcbl64/lib/libFJBASE.so`
 - `/opt/FJSVcbl64/lib/libcobol.so`
 - `/opt/FJSVcbl64/lib/librcobflm64.so`
- if you use shared objects in your application, you must always link them dynamically.
- the C++ linkage editor `CC` if at least one C++ program is contained; in this case, no COBOL programs are allowed.
- If you want to link an application for 64-bit operation, ensure that all the components that you specify during linking are available in 64-bit mode. It is not possible run a mixture of 32-bit mode and 64-bit mode objects.

2.1.5 Linking with a makefile (Unix and Linux systems)

For your application program, you can create a makefile by implementing application-specific changes to the makefile of the sample application supplied with openUTM.

The sample application can be found in the CPIO archive *utm-directory*/CPIO.utmsample following the installation. You can copy the sample program with the *utm-directory*/shsc/install.sample procedure to the user ID under which your UTM application is also to run. Use the `p/config` procedure to create a configuration of the sample application that corresponds to your application. The makefile `makefile` of this sample application can then be modified and extended for your application; see the online documentation for the sample application.

If you want to link to a database, specify the database system used and the required configuration in the `p/config` run. The makefile created by `p/config` then generally contains all the necessary modules and libraries for the database link. The library lists are created under `.liblists/ORACLE` or `.liblists/ORACLECOB`. More details on linking a UTM database application can be found in [chapter “UTM database application”](#).

2.2 Creating application programs on Windows systems

Application programs can be created in C, C++, or COBOL under Windows systems.

2.2.1 Application programs in C and C++ (Windows systems)

For application programs in C and C++, you must work with Microsoft Visual Studio. Carry out the following steps in this case:

1. Set the options in the Visual Studio
2. Create a project for the application
3. Write the programs or modify existing programs
4. Compile and link the application
5. Install the application as a service, if so desired.

This has the advantage under some circumstances that the application can be started and terminated automatically with the system.

The service can also be started and terminated manually if required.

The [section “Setting the options of the Visual Studio \(Windows systems\)”](#) through [section “Compiling and linking the application \(Windows systems\)”](#) describe how to create a statically linked application program.

If you want to load application programs dynamically, you must create these programs as DLLs; see [section “Creating application programs as DLLs \(Windows systems\)”](#).

The [section “COBOL application programs on Windows systems”](#) explains what to note for application programs in COBOL.



The following description applies to Microsoft Visual Studio Version 2010 (English).

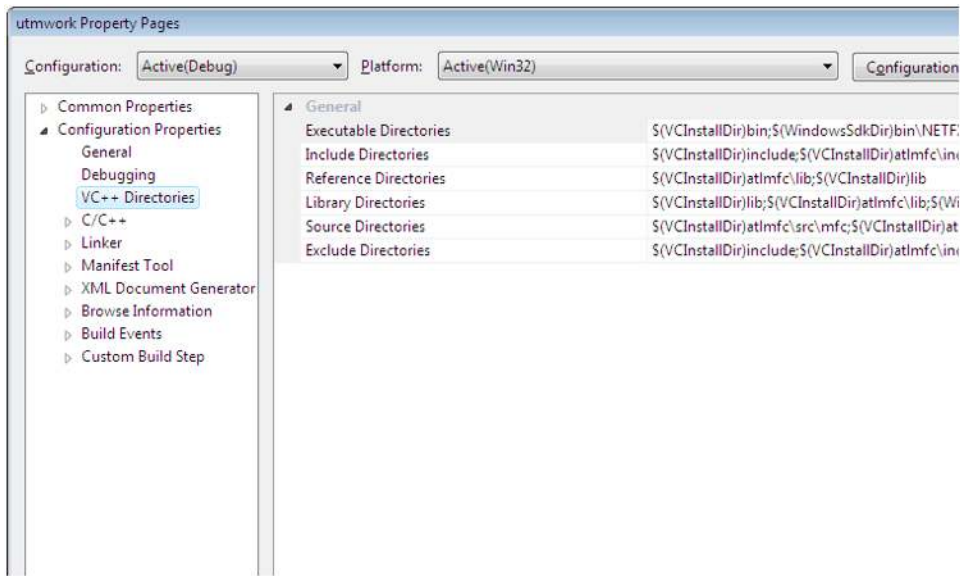
2.2.1.1 Setting the options of the Visual Studio (Windows systems)

Before you can start developing UTM applications, you must add the directories with the openUTM header files (*utmpath* \include), the openUTM library files (*utmpath* \sys) and, if necessary the database libraries, to the development environment.

You set these via certain options of the Developer Studio. These settings are independent of the project and can therefore be used to create various UTM applications.

Proceed as follows:

1. Call the Microsoft Visual Studio, select *Tools - Options* and click *Projects and Solutions*.
2. Select *VC++ Directories* and set the option for the include files as follows:
 - In the *General* for box, select the value *Include Directories* and double-click on the empty data entry line.



- Click the *New Line* button and enter the directory containing the include files: *utmpath* \include
The default is *C:\openUTM-Server\64\include*
Alternatively, click the " ..." button and choose the directory from the intermediate dialog box that follows.
 - Move the directory to the top location using the arrow button.
3. Set the options for the UTM libraries and the database libraries.
To do this, select the value *Library Files* in the *Show Directories for* box and proceed in the same manner as for the header files:
 - a. Specify the directory with the UTM library files. You must always set these options:
 - double-click on the empty input line
 - Enter the directory containing the UTM libraries (*utmpath* \sys) .
The default is *C:\openUTM-Server\64\sys*.
Alternatively, click the " ..." button and choose the directory from the intermediate dialog box that follows.
 - Use the arrow buttons to move the directory to the top,
 - b. Specify the directory with the database library files. This is only necessary when you want to connect a database (for example ORACLE):

-
- double-click on the empty input line
 - Enter the directory containing the database library files directly or select it using the intermediate dialog box (by double-clicking " ... "

4. Now press OK in the *Options* window.

This stores the options for the header files and library files.

2.2.1.2 Creating projects (Windows systems)

Proceed as follows:

1. Call the Visual Studio and select the *File* item in the menu bar, click on *New* and select the *Projects* tab.
2. Highlight *Win32 Console Application* and enter the name of your project in the *Project name* field; the name is freely selectable and is referred to as *utmproject* below.
3. Enter the directory in which your project and all its files are to be stored in *Location*. If you enter a directory here that does not yet exist, then the directory is created.
4. Click on OK. This creates the project. The project remains open.

Opening a project later

If you want to open the project later, then you have two possibilities:

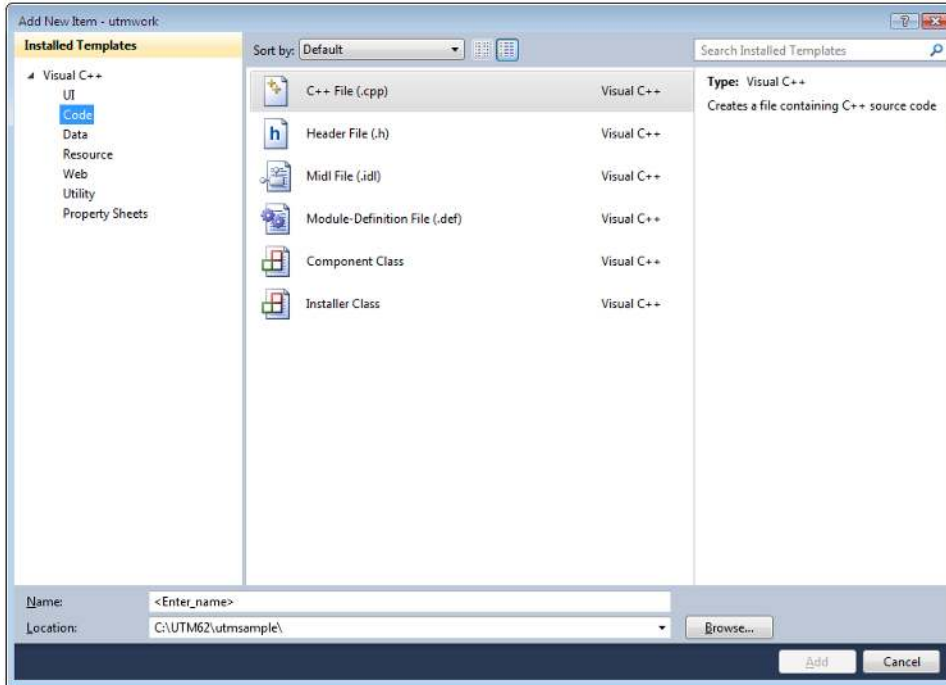
- Via the Visual Studio by clicking on *File - Open - Project/Solution* and by navigating in the application directory with the help of *Search in*, if necessary. Click on *utmproject.sln* (*utmproject* = name of your project) once there.
- Via the Explorer by double-clicking on the *utmproject.sln* file.

2.2.1.3 Writing source programs (Windows systems)

If you want to create new source programs in C or C++, then it is best to use the syntax-sensitive editor of the Visual Studio. You can, of course, create your programs with any common ASCII editor or modify existing programs with the same.

Proceed as follows to create your programs with Visual C++:

1. Open your project by double-clicking on the *utmproject.sln* file in the Explorer, for example. This starts the Visual Studio.
2. Select the *Project* item in the menu bar and click *Add New Item*.



3. In the *Installed Templates* group, select the item *Code*, in the central group, select the item *C++ File (.cpp)* and enter the name of your source program at the bottom of the screen under *Name*.
4. Click on *Add*. This saves the file automatically in the project. The editing window opens.
5. Create the source code in the edit window.
6. Close the file with *File - Close*, thereby saving the changes.

2.2.1.4 Compiling and linking the application (Windows systems)

Requirements

Before you can link the application, you must add all required source programs and object files to your project and set the linker options (see below).

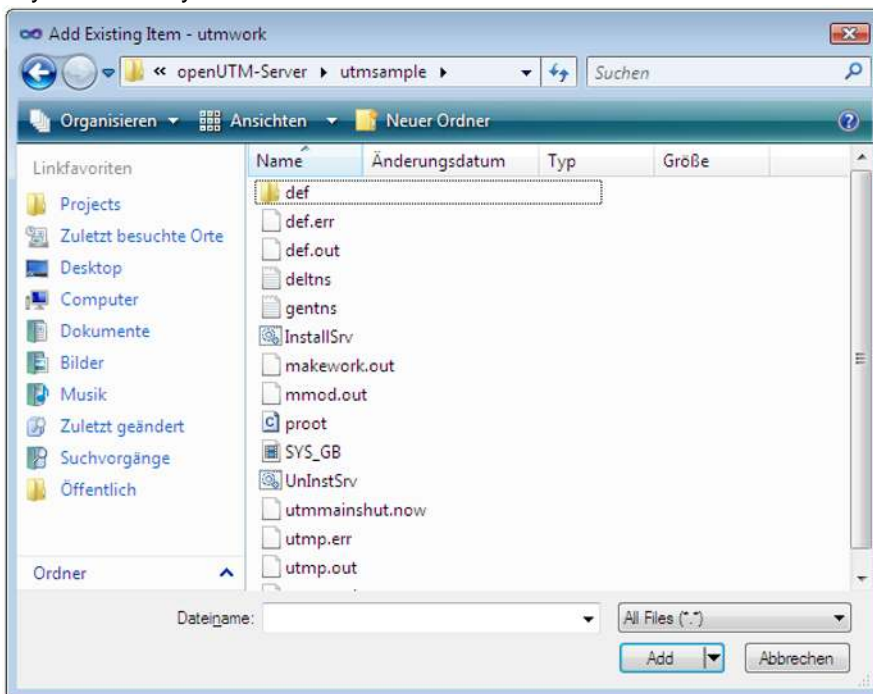
The files `mainutm.c` (to be used if no C++ programs are to be linked) and `mainutm.cpp` (if C++-programs are to be linked) are supplied with openUTM. These sources must be compiled and the resulting objects must be included in the application.

If the UTM utility program KDCMMOD has been used to generate a source for an application-specific message module then this must also be compiled and the resulting object linked into the application.

The ROOT table source that you must always create before linking with KDCDEF also belongs to the required source programs. It is recommended to store all source programs, including the ROOT table source, in the project directory.

Adding source programs and object files to the project

1. Open your project.
2. Add the source programs as follows into your project:
 - a. Click in the menu bar on *Project* and highlight *Add Existing Item...* This opens the *Add Existing Item* window.
 - b. Select the *Visual C++ Files (.c;.cpp;.cxx;.tli;.h;.tlh;.rc)* item as the *File Type*. Navigate in the project directory if necessary.



Highlight the following there:

- The ROOT table source created with KDCDEF, `proot.c` in the example.
- All source programs you have created that are to be linked in the project. Source programs that you have created in your project with the Visual Studio are automatically contained in the project and do not need to be highlighted.

c. Click on

Add. You have now added the source programs to the project.

You can also compile individual source programs beforehand for large projects (*Build - Compile*) and then add the object files as described below instead.

3. Add the UTM object files to your project in a similar manner:

a. Click in the menu bar on *Project* and highlight *Add Existing Item...* The *Add Existing Item* window is opened.

b. Add the UTM object files:

Select the *All Files (*.*)* item as the *File Type*.

Navigate in the directory *utmpath \sys* and highlight the following files:

- Either *mainutm.obj* or the object based on *mainutm.c* if no C++ programs are contained,
- alternatively *mainutmCC.obj* or the object based on *mainutm.cpp* if C++ programs are contained.

mainutm.obj/mainutm.c or *mainutmCC.obj/mainutm.cpp* contain the main function of the application.

c. If it is present, add the application-specific message module - however, this does not necessarily have to be located in *utmpath \sys*.

d. Now click on *Add*. You have added the UTM object files.

4. Adding additional object files:

If you have already compiled programs, then repeat steps a) through c) from 3. for the corresponding object files.

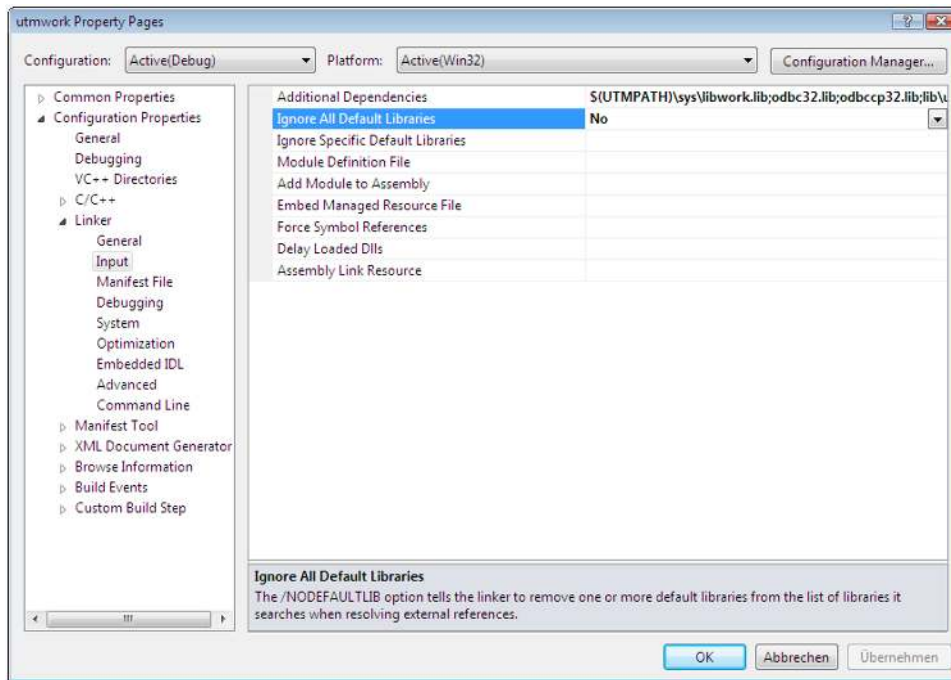
Displaying files

You can view all the files contained in your project at any time by clicking *Solution Explorer* in the navigation area. In the work window, you can, for instance, also open and then edit source programs by double-clicking them.

Setting linker options

You must set the linker options each time before you link an application.

1. Choose *Project - Properties* from the menu to open the *utmwork Property Pages* window of the project (*utmproject*).
2. Choose *Configuration Properties - Linker* in the navigation area.
3. Under *Linker*, click *General* and under *Output File* enter the name *utmwork.exe*. *utmwork.exe* is the name of the linked work process. This name cannot be freely selected.
4. Under *Linker*, click *Input* and under *Additional Dependencies*, enter the UTM library *libwork.lib*. This library must appear **before** all other libraries.



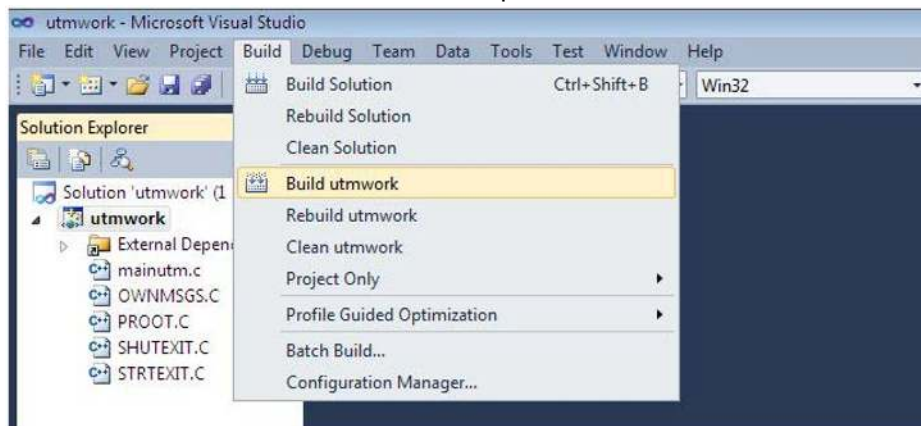
If you use the XATMI interface or databases, then you must also add the following libraries:

- The `libxtc1t.lib` library for XATMI.
- The database library(ies). The library(ies) to be specified can be found in the documentation for the corresponding database.

5. Now click on OK. The linker options you selected are now valid.

Linking the application program

1. Open your project and select x64 for a 64-bit program as the target system.
2. Select the *Build* item in the menu bar and place the cursor on the *Build utmwork* item (see screen snapshot):



As soon as you click on the item, your programs are compiled and your application is linked.

Messages during linking

You receive the messages LNK4075 and LNK4056 when linking. You can ignore these messages; the LNK4056 messages arise because the application programs are source code compatible with Unix and Linux systems.

If you use the `exit()` function in a start exit, then you will also receive message LNK4006. This message is received because the `exit()` function is dealt with by UTM library `libwork.lib` and not by a Windows library as usually. This is the reason why `libwork.lib` must always be the first in the list (see above).

Result of the linker run

The application program `utmwork.exe` is stored in your project directory at the end of the linker run. As the openUTM main process always looks for the application program under this name, you may not rename `utmwork.exe`.

2.2.2 Creating application programs as DLLs (Windows systems)

If you want to create an application program as a DLL, the following differences apply compared to static linking:

1. Create the project as a Dynamic Link Library (see [section “Creating projects\(Windows systems\)”](#)).
2. Add the statement `void declspec(dllexport)` to all program units; see [section “Compiling and linking the application \(Windows systems\)”](#). This can look as follows, for example:

```
void declspec( dllexport ) func (struct kc_ca *kb, struct work *spab)
```

All other steps are the same as for static linking.



DLLs are treated in the same way as shared objects in In Unix and Linux systems. For details on generating and using DLLs (otherwise referred to as shared objects, see [section “Exchanging shared objects”](#)).

2.2.3 COBOL application programs on Windows systems

If you want to create application programs in COBOL, you use the compiler Visual Cobol from Microfocus. COBOL runtime licenses from MicroFocus are required for execution of programs that were compiled with a Micro Focus compiler.

Compiling application programs

Carry out the following steps to compile application programs:

1. Setting environment variables

- > Call the command script `<visualcoboldir>\base\bin\CreateEnv.bat`.
- > Extend the COBCPY environment variable by adding the directory `%UTMPATH%\copy-cobol85`.
- > Extend the INCLUDE environment variable by adding `<path>\include`, where `<path>` is the installation directory of the COBOL compiler (required for the compilation of the root sources).
- > If you create programs based on CPIC, TX or XATMI under openUTM, extend the COBCPY environment variable by `%UTMPATH%\<interface>\copy-cobol85`, where `<interface>` stands for `cpic`, `tx` or `xatmi`.
- > If you create client programs based on UPIC-L, extend the COBCPY environment variable by adding `%UTMPATH%\upicl\copy-cobol85`.

2. Open a command prompt window by choosing *Start - Programs - Command Prompt*, for example, and enter the command `cobol`. You then specify the source files interactively.

You should also compile `%UTMPATH%\src\mfcobol\MAINUTMCOB.cbl` and use the resulting object.

If you have installed the Quick Start Kit, you can also adapt the `workcob.mak` makefile to suit your specific requirements.

Linking application programs

Application programs are linked in two steps:

1. Open a command prompt window and enter the command `cblnames`. In this case you specify all COBOL objects and other objects individually, e.g.

`%UTMPATH%\sys\MAINUTMCOB.OBJ` and `root.obj`.

2. Link the `utmwork.exe` program using the Microsoft linkage program LINK.

The following objects must be integrated:

- `@cbllds.lnk` (output of `cblnames`)
- other application program libraries (if any)
- `%UTMPATH%\sys\libwork.lib` (import library of UTM)
- `cblrtss.lib` (Cobol runtime system)
- C runtime system, e.g. `msvcrt.lib` `kernel32.lib` `user32.lib` `gdi32.lib` `advapi32.lib`



- It may be necessary to set the LIB environment variable to the directories with these libraries.
If the program is to be animated, it is necessary to specify the /BD switch.
To define the COBOL main entry, it is necessary to use the /mMainUtm switch.

The options for both steps can simply be taken from the QuickStart Kit. The makefile for `nmake` is stored in the *filebase* directory under the name `workcob.mak`.



Please also note the compiler-specific programming notes in the manual openUTM manual „Programming Applications with KDCS“ (chapter "Additional Information for Cobol", section "Platform-specific features on Windows systems").

2.2.4 Installing an application as a service (Windows systems)

You can set up an existing UTM application as a Windows service so that the application is started automatically when the computer is booted and terminated automatically when the computer is shut down. You install and uninstall a service by calling the `utmmains` program. `utmmains` is a component of openUTM and is located in `utmpath\ex`.

Installing a UTM service

To install a UTM application as a service you must sign on to Windows system under a user ID with administration rights and open a command prompt window with *Start - Programs - Command Prompt*.

Call `utmmains` with the following parameters in this command prompt window:

```
utmmains [-d] install servicename filebase [ startparam-file [ user]]
```

The parameters must be separated by commas, and the parameters mean the following:

- d
Diagnostics information is also output to *stdout*, optional.
- install
Specifies that `utmmains` installs a service.
- servicename
Variable name part of the service; required parameter. The complete name of the service is then `openUTM servicename`. It is recommended to use the application name from the KDCDEF statement MAX APPLINAME as the *servicename*.
- filebase
Fully qualified base name of the UTM application; required parameter.
The base name is the name of the directory in which the application program and KDCFILE are located.
- startparam-file
Fully qualified name of the start parameter file.
Default: `Startp.std`
- user
Account name for the application run. You can specify a local user or a domain user here. The name must have the following syntax:
For a local user account: `.\LocalUser`
For a domain user account: `DomainName\DomainUser`
Default: system account

If you specify a user here, then you must enter the corresponding password when configuring the service (see below).

`utmmains` sets up the service immediately. You will need to configure the service after that.

Configuring the UTM service

As an administrator you can specify the startup type and the properties of the login account. Proceed as follows to do this:

1. Open the *Control Panel* window by clicking on *Start - Control Panel*.
2. Click on *Administrative Tools* and then on *Services*. This opens a window with a list of all services available on the computer.
3. Highlight the desired service; UTM services always have the prefix `openUTM`. The *Status* column displays if the service has been started or not.
4. Click on the *Startup Type* button. This opens an additional window in which you can specify the following:
 - The startup type (Manual/Automatic/Disabled)
 - The account under which the service is signed on. If you already specified a user when installing the service (*User* parameter), then it is displayed here. In this case **you must** enter the corresponding password here, otherwise you will not be able to start the service.

i If a utility program such as *KDCKAA.exe* cannot access the resources (e.g. shared memory) of the UTM application, it may be because different accounts are involved. If so, you should configure the service so that it signs on with the same account data as the user who starts the utility program.

5. Click on OK and exit the Control Panel.

Deinstalling a UTM service

To deinstall a UTM service, sign on under a user ID with administration privileges, open a *Command Prompt* window and call `utmmains` with the following parameters:

```
utmmains [-d] remove servicename
```

The parameters mean the following:

`d`

Diagnostics information is also output to *stdout*, optional.

`remove`

Specifies that `utmmains` deinstalls a service.

`servicename`

Variable name part of the service.

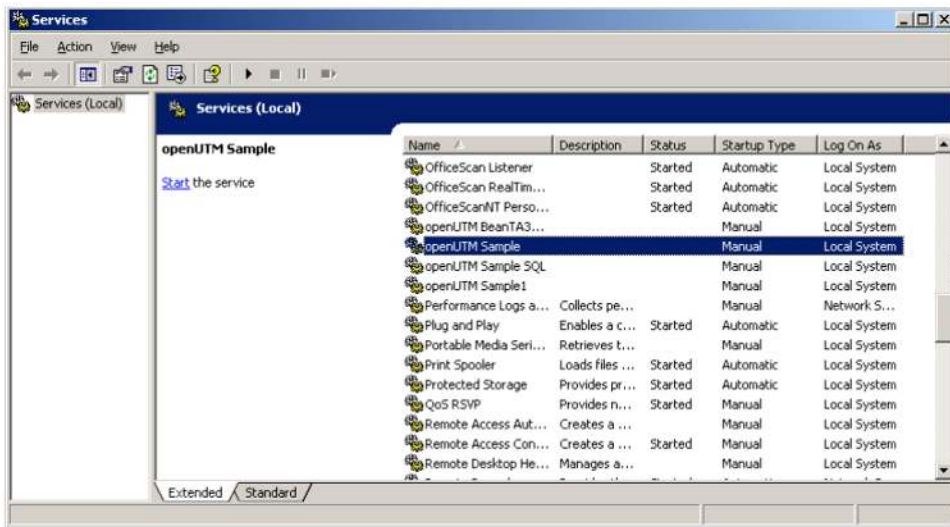
Example

You want to set up the application `SAMplE01` as a service and sign on to the system account. The `KDCFILE`, start parameter file and application program can be found in the directory `C:\utmserve`. The start parameter file is named `startparameter`.

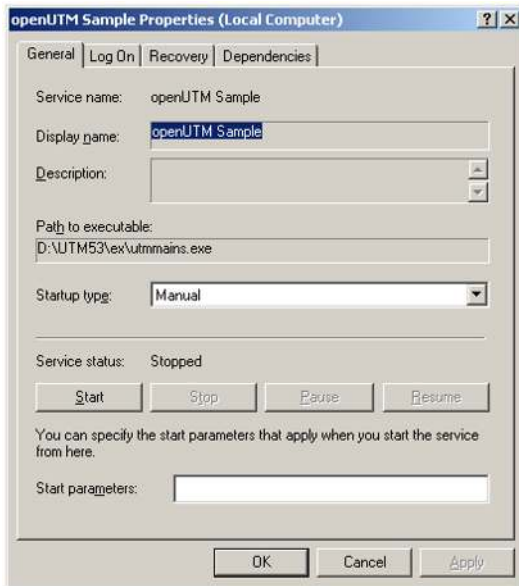
- To install the service, call utmmains as follows:

```
utmmains install Sample C:\utmserv C:\utmserv\startparameter
```

If on Windows you now click on *Start - Control Panel - Administrative Tools - Services* and page down, then you will see a screen like the one in the following figure:



- Click on *General* and configure the service in the following window:



For example, here you can switch to *Automatic* for the *Startuptype* mode.

Please note the following: If you select startup type *Automatic* then the UTM application is not permanently terminated until you terminate this service, see [section "Terminating a service on Windows systems"](#).

- Click on *Log On ...* and configure the service in the following window:



Here you can specify a different account for *This Account* in *Log on as* instead of the system account.

- To deinstall the service, call `utmmains` as follows:

```
utmmains remove Sample
```

3 Necessary files and global system resources

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

- the KDCFILE
- the system files stderr and stdout
- the system log file SYSLOG
- the user log file(s) USLOG (optional)
- the directory DUMP
- all program and object module libraries from which the application is to dynamically load modules during the start phase and during operation.

In standalone applications, KDCFILE, USLOG, SYSLOG and the DUMP directory must be located in the *filebase* directory (= base directory of the UTM application). You must specify *filebase* in the start parameters. You specify the name *filebase* when you create the KDCFILE with the KDCDEF generation tool, see the openUTM manual “Generating Applications”, MAX...,KDCFILE=*filebase* control statement.



Information on what files are required for operating a UTM cluster application is contained in the [chapter “UTM cluster application”](#).

3.1 System files `stderr` and `stdout`

openUTM logs messages to the system files `stderr` and/or `stdout`.

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

Switching system files

The system files can be switched over during live operation either by the administrator or at definable periodic intervals. The system files are always switched over together.

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

The system files are switched over as soon as possible after the request has been made.

- To switch over the system files using a time interval, specify the start parameter `SYSPROT` when starting UTM application (see [section “Start parameter file of the application”](#)). You can specify a time interval in days. The files are always switched over at midnight.

Names of the switched files

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

`stdout: prefix.out.YY-MM-DD.HHMMSS`

`stderr: prefix.err.YY-MM-DD.HHMMSS`

`prefix`

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

Default: `utmp`

`YY-MM-DD.HHMMSS`

Date and time of the switchover

If you set the environment variable `UTM_REDIRECT_FILES` to `YES` (see ["General environment variables for openUTM"](#)), the files are switched over immediately the UTM application is started and output is sent to the files with names corresponding to the formats shown above. Output is not written to the existing system files `stdout` / `stderr` or to their redirection destinations.

3.2 System log file SYSLOG

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

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

The SYSSLOG of the application is always contained in the *filebase* directory, where *filebase* is the directory under which the application is installed (base name of KDCFILE; defined in MAX KDCFILE).

openUTM provides two options for maintaining a SYSLOG:

- as a simple file SYSLOG in the *filebase* base directory
- as a file generation group SYSLOG in the *filebase* base directory

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

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

Messages from openUTM

openUTM outputs the following messages regarding the SYSLOG:

- Message K136 at the start of the application:
`K136 (First) SYSLOG file is &FNAM`
- Message K138 at the end of the application:
`K138 SYSLOG file &FNAM closed`
- Message K137 after switching to another file generation:
`K137 SYSLOG switched to file &FNAM`

3.2.1 SYSLOG as a simple file

You can create the SYSLOG file as a simple file before the start of the application in the *filebase* base directory. If neither a file nor a file generation directory with the name SYSLOG exists in *filebase* when the application starts, then openUTM creates a simple file called SYSLOG.

At the start of the application, openUTM opens the SYSLOG file. It remains open for the entire application run. openUTM writes all the events of an application run into this file.

With each subsequent start of the application, the contents of the SYSLOG file are overwritten by openUTM. The log information from the previous application run is lost. After the end of an application run, you should therefore save the contents of the SYSLOG file, if necessary.

! CAUTION!

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

3.2.2 SYSLOG as a file generation group

openUTM only maintains the SYSLOG as a file generation group if a file generation group called SYSLOG exists in the *filebase* base directory at the start of the application.

You must create this file generation group using the KDCSLOG tool (see also [section “The KDCSLOG tool for creating the SYSLOG-FGG”](#)).

A **File Generation Group** (FGG) is a directory with files that are numbered consecutively using their file names (e.g. 0001, 0002,...). The files are called file generations of the FGG. The numbers are referred to as the file generation numbers.

When the FGG is created with KDCSLOG, the INFO file is also created and written in the FGG. The first time the application starts, openUTM creates the first file generation 0001 in the FGG and opens it as the SYSLOG file. All processes of the application write the messages with the destination SYSLOG in this file generation first.

When switching files, openUTM automatically creates the next generation.

If the file generation with the generation number n is the last file in which openUTM has written before the end of an application run, openUTM creates the file generation $(n+1)$ with the next application start and opens this generation as the SYSLOG file.

The FGG contains a maximum of m file generations. The number m is defined when creating the FGG with the KDCSLOG tool. As soon as openUTM creates the $(m+1)$ -th file generation, the oldest file generation is deleted, i.e. the file generation with the lowest generation number.

3.2.3 The KDCSLOG tool for creating the SYSLOG-FGG

The SYSLOG-FGG is created using the KDCSLOG tool. You will find it in the `ex` subdirectory of the openUTM installation directory. The program is started as follows:

In Unix and Linux systems from a shell with
`utmpath /ex/kdcslog filebase number [K]`

On Windows systems from a command prompt window with
`utmpath \ex\kdcslog filebase number [K]`

Meaning of parameters:

filebase	Name of the directory under which the application is installed or is to be installed (base name of the KDCFILE).
number	Maximum number of file generations in the FGG.
	The FGG contains a maximum of <i>number</i> file generations. As soon as openUTM creates the (<i>number</i> +1)-th file generation, the oldest file generation (i.e. the file generation with the lowest generation number) is deleted.
	Minimum value: 1
	Maximum value: 9999
K	(keep)
	If this parameter is specified, all files are retained even if <i>number</i> is exceeded.

First of all, KDCSLOG creates the *filebase* base directory if it does not yet exist. The FGG SYSLOG is then created in the *filebase* and an INFO file is created within the FGG. The INFO file is used to store all the current status information on the file generations of the group.

! CAUTION!

If a SYSLOG already exists in the *filebase* directory before KDCSLOG is called, this FGG is deleted and a new one created.

KDCSLOG messages

The KDCSLOG tool outputs its messages to *stdout* and *stderr*. The KDCSLOG messages are listed in the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”.

Example

Creating the FGG SYSLOG on Unix and Linux systems

The base name of the application is `/home/userutm/example`. The file generation group for the SYSLOG is created as follows:

```
utmpath /ex/kdcslog /home/userutm/example 10
```

KDCSLOG creates the FGG:

/home/userutm/example/SYSLOG

and the file:

/home/userutm/example/SYSLOG/INFO

Creating the FGG SYSLOG under Windows systems

The base name of the application is `C:\utmsample`. You set up the file generation directory for the SYSLOG as follows:

```
utmpath \ex\kdcsllog C:\utmsample 10
```

KDCSLOG creates the FGG:

`C:\utmsample\SYSLOG`

and the file:

`C:\utmsample\SYSLOG\INFO`

Comments on the examples

The UTM application always writes to the file currently with the highest generation number. If the SYSLOG is switched to the next file generation, openUTM creates this file generation. The maximum possible number of numbered log files is specified in the *number* parameter, i.e. a maximum of 10 file generations. If this number is reached and if the file is switched, the file with the lowest number is deleted, i.e. if openUTM creates file generation 0011 when switching the file generation 0001, the file generation is deleted automatically, and so on.

! CAUTION!

Please make sure that files which have not yet been evaluated are not overwritten or deleted.

3.2.3.1 Automatic size monitoring

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

Automatic size monitoring can be set in two ways:

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

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

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

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

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

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

After the generations have been switched successfully, no more UTM messages are written to the old file generation. When a file generation is closed, openUTM outputs UTM message K138.

3.2.4 Protection against oversized SYSLOG file

If you are maintaining the SYSLOG as an FGG, you can control the amount of storage space occupied by the SYSLOG by permitting a maximum of n file generations for the FGG (*number* parameter of the KDCSLOG tool) and by activating automatic size monitoring. See also [section “Automatic size monitoring”](#).

The file generations of the SYSLOG are cyclically overwritten so that the FGG contains a maximum of n file generations. With size monitoring, each generation has a maximum of *size* UTM pages.

The maximum space requirement of the SYSLOG-FGG is thus calculated by:

$n * size * (\text{size of a UTM page})$.

3.2.5 Behavior in the event of write errors

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

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

- The SYSLOG is maintained as a simple file:

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

- The SYSLOG is maintained as an FGG:

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

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

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

3.3 User log file

The user log file contains the records created by the application program with LPUT calls. The user log files of an application are organized in a file generation group (FGG), i.e. a group of files numbered consecutively using their file names. The user log files are contained in the USLA directory in the *filebase* base directory. If user log files are required (LPUT calls), they must be created using the KDCUSLOG tool before the application starts.

Calling KDCUSLOG

Unix and Linux systems:

```
utmpath/ex/kdcuslog filebase number [ S | D ]
```

On Windows systems from a command prompt window with

```
utmpath\ex\kdcuslog filebase number [ S | D ]
```

Meaning of parameters:

filebase	Name of the directory under which the application is installed or is to be installed: Base of the KDCFILE.
number	Number of files in each file generation group; maximum 9999.
S	Single-file operation; default setting.
D	Dual-file operation; the USLA directory is also created in the <i>filebase</i> base directory.

KDCUSLOG first of all creates the *filebase* directory if it does not already exist. The USLA directory and, for dual-file operation, the USLB directory are then created in *filebase*. An INFO file which is used to store the current status information on files in the FGG is created in the USLA or USLB directory in which the current status information on files of the FGG are stored.

The USLA directory contains the following files:

/INFO: Administrative file

/0001: First file of the file generation (number 0001)

! CAUTION!

If the file generation group already exists before the procedure is called, the old group is deleted and a new one created.

Example

Unix and Linux systems

The base name of the application is `/home/userutm/example`. The user log file is to be operated in dual-file mode.

The file generation group for the user log file is created as follows:

```
utmpath/ex/kdcuslog /home/userutm/example 2 D
```

KDCUSLOG then creates the files:

```
/home/userutm/example/USLA/INFO
/home/userutm/example/USLA/0001
/home/userutm/example/USLB/INFO
/home/userutm/example/USLB/0001
```

Windows systems

The base name of the application is `C:\utmsample`. The user log file is to be operated in dual-file mode.

The file generation group for the user log file is created as follows:

```
utmpath\ex\kdcuslog C:\utmsample 2 D
```

KDCUSLOG then creates the files:

```
C:\utmsample\USLA\INFO
C:\utmsample\USLA\0001
C:\utmsample\USLB\INFO
C:\utmsample\USLB\0001
```

Comments on the example

The UTM application always writes to the file currently with the highest number. With each KDCLOG command issued by the administrator, openUTM switches to the next file generation. The maximum number of numbered user log files is specified in the *number* parameter (in Example 2) when KDCUSLOG is called. If this number is reached and if the generation is switched with KDCLOG, the file with the lowest number is deleted.

Make sure that files which have not yet been evaluated are not overwritten.

openUTM does not write the user log records directly into the log file, rather saves them first of all in the page pool of the KDCFILE. If the page pool contains the number of UTM pages generated in `MAX...,LPUTBUF=number`, openUTM copies the records to the user log file. The records are copied asynchronously to active transactions. If the application is terminated normally, openUTM likewise copies the records to the user log file.

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

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

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

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

KDCUSLOG messages

KDCUSLOG outputs its messages to *stdout* and *stderr*. The KDCUSLOG messages are listed in the openUTM manual „Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems“.

3.3.1 Response to write errors

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

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

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

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

3.4 DUMP directory

The following files are stored in the DUMP directory:

- dump files that were possibly created during the application run
- temporary files needed to create the dump
- if necessary, core files for diagnostics (Unix and Linux systems) or mini dumps (Windows systems)

This directory should therefore always be set up before startup, so that these files can be created. The DUMP directory must be created in the *filebase* base directory.

3.5 Global system resources of an application

The global system resources that a openUTM production application requires are listed in this section. You will learn how to change the size of the shared memory area for interprocess communication (IPC) to improve the performance of your application when communicating with network partners.

3.5.1 System resources required by a UTM application

An openUTM production application requires the following global system resources.

Shared memory area

A UTM application requires three shared memory areas for the configuration data and global application administrative data (KAA), the cache, and the internal UTM process communication (see also [section “Improving performance: Changing the size of the dataarea in the IPC shared memory”](#)).

For communication via OSI TP, OSS and XAPTP shared memory are also required.

Semaphores

A UTM application requires semaphores for controlling and synchronizing processes of the UTM application.

In openUTM, the semaphores are organized as semaphore arrays where each semaphore array contains exactly 20 semaphore entries. A semaphore array contains one or more semaphores. A UTM application requires at least one semaphore array. The maximum number of semaphores in the system is limited.

In the openUTM environment, the semaphore entries are allocated as follows:

- nine entries for IPC shared memory
- one entry for KAA shared memory
- one entry for CACHE shared memory
- one entry for CACHE access lock
- one entry for PCMM access lock
- one entry for OSS and one entry for XAPTP shared memory
- one entry for each work process as a task bourse
- one entry for each attached external process (*utmtimer*, *utmdtp*, *utmprint* and local UTM client program) for communication between work process and the external process.
- two entries for each connected network process of type *utmnet*, *utmnets* or *utmnetssl*, to permit communication between work process and the network processes.

The number of network processes started depends on the type of network connection of a UTM application:

Connection via PCMX

One process (*utmnet*) is started for each listener ID generated with KDCDEF (BCAMAPPL or ACCESS-POINT statement).

Connection via the socket interface (native TCP/IP)

One socket network process (*utmnets*) is started for each Socket listener ID generated with KDCDEF (BCAMAPPL statement with T-PROT=SOCKET and SECURE=NO).

One TLS socket network process (*utmnetssl*) is started for each Socket listener ID generated with KDCDEF (BCAMAPPL statement with T-PROT=SOCKET and SECURE=YES).

This means:

At least one dialog terminal process (*utmdtp*) can be connected for a minimum production application (single-process application) for which one key for the semaphores is generated.

With MAX...,SEMARRAY= you can define a range of up to 1000 sequential keys. KDCDEF generation with MAX...,SEMKEY= allows you to define up to 10 separate keys for semaphores.

i The message U189 with shortage "SEMA USED" is output and the UTM application is terminated abnormally if you specify too few semaphore keys in the UTM generation with KDCDEF.

File descriptors

A work process of a UTM application always allocates the following file descriptors for:

- *stdin*
- *stdout*
- *stderr*
- the KDCFILE file
- the SYSLOG file
- the IPC shared memory
- the KAA shared memory
- the CACHE shared memory
- a named pipe to the main process (*utmmain*)
- a named pipe to the logging process (*utmlog*)

Additional file descriptors for communication via OSI TP for:

- the OSS shared memory
- the XAPTP shared memory

Further file descriptors are required if dual-file operation is implemented for the KDCFILE, if PAGEPOOL files are used (specification for MAX ...,PGPOOLFS=), or if the restart area is divided into a number of files (specification for MAX ...,RECBUFFS=).

In UTM cluster applications, additional file descriptors are required for the files that are global to the cluster:

- Cluster configuration file
- Cluster user file
- Cluster page pool administration file
- Cluster page pool file(s)
- Cluster GSSB file
- Cluster ULS file
- Cluster administration Journal
- Cluster lock file
- Cluster start serialization file

A work process of the application briefly allocates further file descriptors for:

- the current user log file (USLOG)
- dump files in the event of errors
- startup of a node application in a UTM cluster application

The applifile

This file is created in the installation directory when openUTM is installed and contains the names of all applications started in the system since then, along with their status information and keys of the semaphore and shared memory segments used for communication between the external processes (dialog terminal, printer and timer process and local UPIC client programs) and the work processes. The keys must be assigned uniquely throughout the entire system.

! CAUTION!

The applifile is an internal UTM administration file. You may **not** open this file with an editor. You may destroy the applifile if you do.

3.5.2 Improving performance: Changing the size of the data area in the IPC shared memory

The shared memory area for interprocess communication (IPC shared memory area) requires the UTM application for the exchange of messages between its processes. It is created by openUTM. If this area is too small, then performance bottlenecks can arise and connections may be cleared.

Most of the IPC shared memory area is used to store the messages that are exchanged between the processes of an application. This area will be called the data area in the following discussion. The rest of the shared memory is used to administer the processes and their connections.

The data area in IPC shared memory is organized in 4 KB blocks.

The size of the IPC shared memory is set by openUTM for each application. The size is mainly determined by the number of communication partners generated and by the number of semaphores generated. See also [“Semaphores” \(System resources required by a UTM application\)](#).

openUTM creates a data area of approximately 10 * 4 KB for each semaphore key generated. In addition, openUTM creates a data area of approximately 4 KB for each communication partner generated.

The data area created by openUTM may be too small if a lot of data is sent over the connection to the communication partners. This can lead to performance bottlenecks and therefore to the clearing of connection. To prevent this, you can change the size of the IPC shared memory. The environment variables UTM_IPC_LETTER and UTM_IPC_EXTP_LETTER are used to do this. You can change the absolute size of the IPC shared memory with UTM_IPC_LETTER, and with UTM_IPC_EXTP_LETTER you can change the maximum size of the data area in IPC shared memory that is available for a single connection.

Changing the absolute size of the data area

The data area in IPC shared memory is distributed amongst the existing connections according to the “first come - first served” principle. If the entire data area is in use, then connections are cleared.

openUTM then outputs the following message:

```
U189 IPC bottleneck &IPCOBJ &IPCREAS
```

with the inserts &IPCOBJ=LETT and &IPCREAS=IPC FULL or EXTP FULL.

You can increase the absolute size of the data area using the environment variable UTM_IPC_LETTER to prevent this. In UTM_IPC_LETTER you specify the number of 4KB blocks that are to comprise the IPC shared memory. The smallest value allowed is 5 (corresponding to 20KB).

A change to UTM_IPC_LETTER will only take effect after the next start of the UTM application. UTM_IPC_LETTER is evaluated at the start of the application by the first work process.

If you have very little data, then you can also decrease the size of the data area with UTM_IPC_LETTER. You can then reduce the amount of overhead required by the operating system to administer the IPC shared memory.

Changing the maximum data area for the messages produced by a connection

All connections are basically handled in the same manner when allocating space in the data area in IPC shared memory. To prevent the data area from being used up from just one connection, there is a default maximum size of 64 KB (16 * 4KB) that can be used by one connection at any one time. If all 64 KB are already in use by one connection, then the connection is cleared.

openUTM then outputs the following message:

U189 IPC bottleneck &IPCOBJ &IPCREAS

with the inserts &&IPCOBJ=LETT and &IPCREAS=MAX ILETT or MAX OLETT.

You can increase the maximum size of the data area available for a connection using the environment variable UTM_IPC_EXTP_LETTER to prevent this.

With UTM_IPC_EXTP_LETTER you can specify the maximum size of the data area available for a connection in 4KB blocks. The default value is 16. The smallest value allowed is 1 (corresponding to 4KB).

A change to UTM_IPC_EXTP_LETTER will only take effect after the next start of the UTM application. UTM_IPC_EXTP_LETTER is evaluated at the start of the application by the first work process.

4 Starting a UTM application

A UTM application is started by calling the *utmmain* program. This program is the main function of the application and runs as a background process. *utmmain* creates the work processes (*utmwork*), the timer process (*utmtimer*) and, if required, network processes (*utmnet*, *utmnets*, *utmnetssl*) and printer processes (*utmprint*, Unix and Linux systems only). See also the openUTM manual “Concepts und Functions”.

A UTM application can also be started with a debugger for diagnostic purposes (see openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”, section “Debugging UTM applications”).

Please note that the *utmmain* program must always be started under the application directory (*filebase*).



For information on cluster-specific issues when starting a UTM cluster application, refer to the [section “Starting a UTM cluster application”](#).

4.1 Starting a UTM application on Unix and Linux systems

You need to take the following steps to start an application with *utmmain*:

1. Set the `UTMPATH` environment variable to *utmpath*. It makes sense to add the following commands to the `.profile` or in the *login-file* of the login name under which the application will be run:

```
UTMPATH=utmpath
export UTMPATH
```

For more information on *utmpath* see "[Installing UTM system functions on Unix and Linux systems](#)".

2. Create the start parameter file as described on "[Start parameters for openUTM](#)".
3. Start *utmmain* in the *filebase* directory as a background process:

```
utmpath/ex/utmmain filebase [ startparam-file] &
```

filebase

is the fully qualified base name for the UTM application (the name of the directory of the application).

startparam-file

is the fully qualified name of the file in which the start parameters are defined. If this parameter is omitted, then the start parameters must be in the file *filebase/startparameter*.

utmmain produces messages on *stdout* at the start of the application as well as during the application run; error messages occurring at the start are output to *stderr* (see also "[Error messages at the application start](#)"). These messages can be redirected to a file or a filter program as in the following example.

Example

The application and the start parameter file are located in the directory `/home/utmbsp`, the start parameter file has the default name `startparameter`.

If all output is to be redirected to a file, then call *utmmain* as follows:

```
utmpath/ex/utmmain /home/utmbsp 1>utmp.out 2>utmp.err &
```

If the output is to be handled further by a program or a shell script named *filter* instead, then call *utmmain* as follows, for example:

```
utmpath/ex/utmmain /home/utmbsp 2>&1 | filter
```

Using this method, you can react to messages from the network process, for example, which is not possible with an MSGTAC program.

i Redirection to the *filter* script is only possible if automatic switchover of the log files has not been activated (see SYSPROT in [section "Start parameter file of the application"](#)).

Network processes

When *utmmain* is started, it generates the following network processes.

- One or more *utmnet* processes.
- One or more *utmnets* resp. *utmnetsssl* for TLS communication socket network processes are also started for TCP/IP communication, see also [section "Using different socket network processes \(Unix and Linux systems\)"](#).

4.2 Starting a UTM application on Windows systems

You can start a UTM application with the *utmmain* program or as a service under Windows systems.

4.2.1 Starting with utmmain (Windows systems)

You need to take the following steps to start an application with *utmmain*:

1. Add the *utmpath* \ex directory to the PATH variable for the user ID under which the UTM application is to be run.
2. Create the start parameter file as described on ["Start parameter file of the application"](#).
3. Open a command prompt window with *Start - Programs - Command Prompt*, change to the application directory, and enter the following at the prompt:

```
utmmain filebase [startparam-file]
```

filebase

is the fully qualified base name for the UTM application (the name of the directory of the application). You can also enter *filebase* as a relative path name, e.g. as "." (dot) if you call *utmmain* from the *filebase* directory.

startparam-file

is the fully qualified name of the file in which the start parameters are defined. If this parameter is omitted, then the start parameters must be in the file *filebase* /startparameter.

You can also create a shortcut for the *utmmain* call so that you can start the UTM application with the mouse or with a hot key. See the following example for more details.

utmmain produces messages on *stdout* at the start of the application as well as during the application run; error messages occurring at the start are output to *stderr* (see also ["Error messages at the application start"](#)). These messages can be redirected to a file as shown in the following example.

Example

The application is located in the directory `C:\utmtest\example` and you have added the directory *utmpath* \ex to the PATH variable of your user ID. The start parameter file has the default name *startparameter* and is also located in the directory of the application.

If you want to redirect all messages to a file, then open a command prompt window now and start the UTM application as follows:

```
cd C:\utmtest\example
```

```
utmmain . 1>utmp.out 2>utmp.err
```

If the output is to be handled further by a program or a command file named *filter* instead, then call *utmmain* as follows:

```
utmmain . 2>&1 | filter
```

Using this method, you can react to messages from the network process, for example, which is not possible with an MSGTAC program.

i Redirection to the *filter* script is only possible if automatic switchover of the log files has not been activated (see SYSPROT in [section "Start parameter file of the application"](#)).

Starting the UTM application using a shortcut

You can create a shortcut for this purpose so that you can start the application with the mouse or using a certain keyboard command.

On Windows systems proceed as follows:

1. Click on an empty part of the screen background with the right mouse key, select *New* and click on *Shortcut*. The *Create Shortcut* window opens:
 - Enter the following in the *Location of the item* field:

```
cmd.exe /c utmmain . ./startp.std >utmp.out 2>utmp.err <nul
```

A command prompt window is opened by `CMD` and the command is then executed there. `/C` means that the window is to be closed after `utmmain` terminates.
 - Click *Next* and specify an appropriate name in the *Create Link* window, e.g. `start-utm`.
 - Click on *Finish*. The window closes and an icon with the name `start-utm` appears on the screen.
2. Click on the icon with the right mouse button and select *Properties*. Click on the *Shortcut* tab and execute the following steps there:
 - Enter the directory `C:\utmtest\example` (= application directory) in the *Start in* field. This makes `utmmain` look for the parameters in this directory, and it also stores the files in this directory.
 - Place the cursor in *Shortcut Key* field and press `CTRL`, `ALT` and `F3` at the same time. You can also assign a different icon to the shortcut in this window via the *Change Icon...* button.
 - Press `OK`. The shortcut is now done, and the application can be started by double-clicking on the icon or by pressing `CTRL+ALT+F3`.

4.2.2 Starting as a service (Windows systems)

A UTM application must be installed and configured as a service as described in [section “Installing an application as a service \(Windows systems\)”](#). You can set the startup type to *Automatic* when you are doing this so that the service is started every time the computer is booted. If the startup type is set to *Manual*, then the service must always be started manually.

On Windows, for instance, you start a service as follows:

1. Sign on under a Windows system user ID that has administration privileges.
2. Call up the control panel with *Start - Control Panel*.
3. Click on *System and Security - Administrative Tools* and then on *Services* and highlight the desired UTM service using the right mouse button; the service is always named `openUTM servicename`. The *servicename* is assigned when the service is installed.
4. Select the *Start* command in the context menu. The service is started.

An application started as a service produces the same messages when started and during operation as an application started via *utmmain*. Default, these messages are written to the following files:

- messages to *stdout* in the file `filebase\utmp.out`
- messages to *stderr* in the file `filebase\utmp.err` (see also ["Error messages at the application start"](#)).

The file name depends on the start parameter specified for SYSPROT and whether or not automatic switchover has been specified for the system files (see [section “Start parameterfile of the application”](#)).

i If the application is started as a service and the system account is used (default setting), a number of diagnostic files may be saved in the system directory .

4.3 Starting a UTM application with OpenSSL

In UTM applications the user must provide an openssl library for the encryption functionality.

Both when starting an utmwork process and when starting the KDCDEF utility, an attempt is made to load the openssl library.

Only after successful loading of a suitable openssl library is the encryption functionality available in the UTM application resp. KDCDEF utility.

The result of this loading of the openssl library is output to stderr via a K078 message.

The openssl library is assigned via the environment variable UTM_SSL_LIBRARY. If you do not explicitly set the environment variable, a default name is used. See the [Environment variables for using openssl](#) section for details.

Error behaviour

If the openssl library cannot be successfully loaded, the encryption functionality is not available in the UTM application. However, this has no effect on the remaining functionality of openUTM.

Version of openssl library

Details about the version of the openssl library can be found in the release note for openUTM.

4.4 Starting a UTM application with TLS connections

In addition to the encryption functionality, the user must also provide an openSSL library for TLS connections.

If BCAMAPPLs with T-PROT=(SOCKET,....,SECURE) are generated for a UTM application, a number of network processes of type *utmnetssl* are started when the application is started.

In each *utmnetssl* process, communication takes place via the Transport Layer Security (TLS).

When an *utmnetssl* process is started, the TLS environment is initialized. In addition to loading the openSSL library, the certificates and private keys required for communication are also loaded.

The following user actions are required for a successful start of a *utmnetssl* process:

1. Providing a suitable openSSL library
2. Setting the environment variable UTM_SSL_LIBRARY
For details, see chapter [Environment variables for using openSSL](#).
3. Create a UTM TLS configuration file under the <filebase> directory.

Error behaviour

If the initialization of the TLS environment is not successful, the *utmnetssl* process is terminated and afterwards the UTM application is also terminated abnormally.

Version of openSSL library

Details about the version of the openSSL library can be found in the release note for openUTM.

UTM TLS configuration file

In the UTM TLS configuration file, you specify options for TLS communication.

These are the file names for the file with the SSL server certificate and the file with the private SSL server key.

The content of the UTM TLS configuration file is evaluated at the start of each *utmnetssl* process and the certificates /keys specified there are loaded and checked.

In case of an error, the *utmnetssl* process is terminated.

File name

The name for the UTM TLS configuration file is fixed:

```
utm.ssl.conf
```

File format

The file is line-oriented. The first line contains a header, followed by the options. Empty lines and comment lines are also allowed.

The header - first line - has the following format:

```
#@(#) openUTM SSL Config File
```

Comment lines starts with # or *.

The specification for the certificate and the key is made in the following form

```
RSACertificateFile=<filename>
```

resp.

```
RSAKeyFile=<filename>
```

RSACertificateFile

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

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

RSAKeyFile

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

The following maximum values and rules apply to the UTM TLS configuration file:

- The first line must be the header
- Maximum line length: 300 characters
- Blank lines and comment lines are ignored.

i Example UTM TLS configuration file

A sample file for a UTM TLS configuration file is delivered under the utmpath/ssl directory.

4.5 Start parameter file of the application

The start parameter file is created by the administrator of the application using any editor.

The start parameters define the current runtime parameters of the application. This includes the number of work processes with which the application is to work or possibly parameters for a Resource Manager, for example.

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

- Start parameters with the prefix “.UTM” or without a prefix are interpreted by openUTM itself.
- Start parameters with the prefix “.RMXA” are forwarded by openUTM to the connected Resource Manager (database system) for evaluation (see ["Start parameters for a UTM database application"](#)).

The sequence of start parameters for openUTM and the database system is arbitrary, although the input of all start parameters must be concluded by the END command.

Comments

All lines with an asterisk (*) or hash character (#) in column 1 are interpreted as comments. Comments can be placed anywhere in the start parameter file. You can then activate or deactivate individual start parameters, for example, depending on the application run.

4.5.1 Start parameters for openUTM

The syntax of the UTM start parameters is illustrated below:

```
[.UTM] START {FILEBASE= filebase | CLUSTER-FILEBASE= cluster_filebase }
[ ,ADMI-TRACE= ON | OFF ]
[ ,ASYNTASKS= number ]
[ ,BTRACE={ { ON | OFF } | ( { ON | OFF }, length ) } ]
[ ,CPIC-TRACE = { TRACE | BUFFER | DUMP | ALL | OFF } ]
[ ,DB-CONNECT-TIME= sec ]

[ ,DUMP-CONTENT={ STANDARD | EXTENDED } ]
[ ,DUMP-MESSAGE=( event-typ , event ) ]
[ ,NODE-TO-RECOVER= node-name ]

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

[ ,RESET-PTC = { YES | NO } ]
[ ,STXIT={ ON | OFF } ]
[ ,SYSPROT=( interval , filename-prefix ) ]
[ ,TASKS= number ]
[ ,TASKS-IN-PGWT= number ]

[ ,TESTMODE={ ON | OFF | FILE } ]

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

[.UTM] END
```

In the syntax above the parameters are specified in a line **without a carriage return** and are separated by commas.

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

Syntax check at the start of the application

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

Meaning of the commands

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

END This command concludes the input of start parameters.

Meaning of the operands

FILEBASE=filebase

The base name for the KDCFILE and the user log file in standalone applications. Here you must specify the name under which the KDCFILE is stored at startup time. If an invalid name is specified, the application start is canceled. *filebase* may be a maximum of 29 characters in length, irrespective of whether the name is specified as fully or partially qualified.

If you specify the start parameter FILEBASE, you must not specify the start parameter CLUSTER-FILEBASE.

In the case of UTM cluster applications, use the start parameter CLUSTER-FILEBASE instead of FILEBASE. The base name of an individual node application is defined in the CLUSTER-NODE statement during UTM generation.

CLUSTER-FILEBASE=cluster_filebase

If you want to start a UTM application as a node application of a UTM cluster application, you use this start parameter to specify the basename for the cluster files. Here you must specify the name under which the files that are global to the cluster are stored at the startup time.

CLUSTER-FILEBASE applies locally to the node.

If you specify an invalid name here, the application start is canceled. For information on what names are valid, refer to the openUTM manual "Generating Applications".

If you specify the start parameter CLUSTER-FILEBASE, you must not specify the start parameter FILEBASE.

ADMI-TRACE=

Enables/disables the ADMI trace function (= trace function for the KDCADMIN administration program interface), see also openUTM manual "Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems".

In UTM cluster applications, ADMI-TRACE applies locally to the node.

For information on the names of the trace files, see "[Trace files](#)".

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

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

Default: OFF

ASYNTASKS=number

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

In UTM cluster applications, ASYNTASKS applies locally to the node.

Default: the number specified in MAX...,ASYNTASKS=*number*.

Minimum value: 0

Maximum value: the number specified in MAX...,ASYNTASKS=*number*.

BTRACE= Enable/disable the BCAM trace function.

In UTM cluster applications, BTRACE applies globally to the cluster.

ON The BCAM trace function is enabled at the start of the application. All events relating to the connection are recorded in the BCAM trace file. The description of the BCAM trace file and its analysis using the utility program KDCBTRC can be found in openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”.

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

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

Default: 256
Minimum value: 32
Maximum value: 32767

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

CPIC-TRACE=

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

In UTM cluster applications, CPIC-TRACE applies locally to the node.

For information on the names of the trace files, see [“Trace files”](#).

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

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

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

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

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

DB-CONNECT-TIME=sec

Maximum time in seconds the system waits to establish a connection to the database. If no connection is established to the database during this wait time, message K078 is issued and the utmwork process is terminated.

In UTM cluster applications, DB-CONNECT-TIME applies locally to the node.

Default: 0 (no timeout)

Minimum value: 60

Maximum value: 3600

DUMP-CONTENT=

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

In UTM cluster applications, DUMP-CONTENT applies locally to the node.


STANDARD

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

Default value: STANDARD

EXTENDED

The global process storage areas are contained in all dumps of a DUMP file generation.

 This value should only be set if explicitly requested by the Service personnel.

DUMP-MESSAGE= (event-type, event)

Event where UTM generates a UTM dump with identifier MSGDMP when test mode is enabled. A dump is only created by the process in which the event occurred; the application is not terminated in the process.

In UTM cluster applications, DUMP-MESSAGE applies globally to the cluster.

The dump code depends on the event:

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

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

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

The UTM dump is created when message Knnn is output.

In the case of message numbers K023, K043, K061, K062, a dump is only created once, after which *event-code* is reset automatically.

With all other messages, a dump is created each time the message number occurs until the value is reset by administration.

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

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

Specify a KDCS return code (KCRCCC, e.g. 40Z) for rccc. If this return code occurs with a KDCS call, the UTM dump with the code CC- 40Z is created by the process in which the return code occurred. The message dump for this event is then automatically deactivated.

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

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

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

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

Notes:

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

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

NODE-TO-RECOVER=node-name

This parameter is only relevant for UTM cluster applications.

node-name is the name of the node application for which a node recovery is to be performed. The name results from the UTM generation, see openUTM manual “Generating Applications”, CLUSTER-NODE statement, NODE-NAME operand. Whenever a node application starts, terminates or fails, the K169 message outputs *node-name* together with the host name. WinAdmin /WebAdmin also display the *node-name* in the list of cluster nodes.

Node recovery should only be performed for an abnormally terminated node application if a normal node warm start is either not possible or cannot be performed quickly because the node computer has failed and no virtual host has been defined. As a result, a node recovery for a node application is only possible on a node computer on which the abnormally terminated node application has not run.

For information on the conditions that must be fulfilled in order to perform node recovery for UTM cluster applications as well as on the purpose and function of node recovery, see [section “Node recovery”](#).

If a database system does not support node recovery then node recovery always terminates abnormally.

Default: Blanks, i.e. normal application start.

OTRACE=

Switches on/off the OSS trace function on the start of the application.

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

In UTM cluster applications, OTRACE applies globally to the cluster.

ON Switches on the OSS trace function.

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

(SPI, INT, OSS, SERV, PROT)

Switches on the OSS trace function on the start of the application. Trace records of the specified type are logged. The trace records are specified in an arbitrary sequence.

SPI The XAP-TP system programming interface is logged.

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

OSS The OSS calls are logged.

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

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

OFF The OSS trace function remains deactivated on the start of the application.

Default: OFF

RESET-PTC =

This parameter is only relevant for UTM cluster applications if a value other than blanks has been set for NODE-TO-RECOVER.

RESET-PTC specifies whether transactions with the state PTC ("prepare to commit") are rolled back during node recovery.

A transaction with the PTC state may contain locks on global UTM storage areas that apply globally throughout the cluster and may possibly impair the current UTM cluster application.

Transactions with the PTC state cannot be committed on a node recovery because no connections are established to partner applications. If transactions remain in the PTC state then the node recovery terminates abnormally, i.e. no online import or KDCUPD with the KDCFILE of the failed node application is permitted and any locks on UTM storage areas that are effective throughout the cluster are retained.

If you are able to tolerate possible data inconsistencies, repeat the node recovery with RESET-PTC=YES in the case of existing transactions in the PTC state.

YES Transactions in PTC state are rolled back on node recovery.

NO Transactions in PTC state are retained on node recovery.

Default: NO

STXIT=

Activates signal handling in openUTM.

In UTM cluster applications, STXIT applies locally to the node.

ON Signal handling is activated in openUTM.

With STXIT=ON, the functionality „User Signal Routine“ can also be used on Unix and Linux systems.

Default value: ON

OFF Default error handling for signals remains deactivated.

i *Unix and Linux systems:*

- STXIT=OFF on Unix and Linux systems results in additional diagnostic memory dumps (gcore dumps) in the base directory of the application each time a work process is terminated. The gcore dumps are only written if the *gcore* program exists in the */bin* directory.

You are notified about the gcore dumps in UTM message K078:

```
K078 STXIT OFF in utmwork: termination of utmwork process  
creates gcore-dump
```

- With STXIT=OFF, the functionality „User Signal Routine“ cannot be used on Unix and Linux systems.

SYSROT= Switch over the system files *stderr* and *stdout*.

interval Switchover interval in days.

In UTM cluster applications, interval applies globally to the cluster.

Default: 0 (no interval, files are switched over using administration facilities)

Maximum value: 364

filename-prefix

Prefix for the new file name of the system files that have been switched over. The prefix can be a fully-qualified or partially-qualified part of the file name.

In UTM cluster applications, filename-prefix applies locally to the node.

Default: utmp

Maximum length: 31 characters

You will find a comprehensive description of switching over the system log files in the [section "System files stderr and stdout"](#).

TASKS=number

Number of work processes that are to work for this application.

In UTM cluster applications, TASKS-IN-PGWT applies locally to the node.

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

Minimum value: 1 *)

Maximum value: Number defined in MAX...,TASKS=number

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

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

TASKS-IN-PGWT=number

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

In UTM cluster applications, TASKS-IN-PGWT applies locally to the node.

Default value: Number defined in MAX ...,TASKS-IN-PGWT=number

Minimum value: 1 if MAX...,TASKS-IN-PGWT > 0; otherwise 0.

Maximum value: Number defined in MAX ...,TASKS-IN-PGWT=number

TESTMODE= Activate test mode.

See also the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”, chapter “Debugging and error diagnosis”.

In UTM cluster applications, TESTMODE applies globally to the cluster.

ON Test mode is to be switched on when the application starts. In test mode, additional internal UTM plausibility checks are carried out, and trace information is logged in the internal trace area. Test mode should only be switched on to diagnose UTM errors on the recommendation of the systems analyst.

i With MAX...,IPCTRACE= (see openUTM manual “Generating Applications”, MAX statement), the number of trace information entries written with TESTMODE=ON can be specified in the KDCDEF generation. IPCTRACE should only be defined for diagnosing serious UTM errors on the recommendation of the systems analyst.

OFF Test mode is to remain deactivated when the application starts.

Default value: OFF

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

filebase.KTATRC.pid (*pid* max. 4-position)

TX-TRACE=

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

In UTM cluster applications, TX-TRACE applies locally to the node.

For information on the names of the trace files, see “Trace files”.

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

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

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

DEBUG	The TX trace function is enabled with the level DEBUG at the start of the application. The level DEBUG includes the level FULL, and diagnostic information is also logged.
OFF	The XATMI interface trace function remains disabled at the start of the application. Default: OFF
XATMI-TRACE=	
	Enables/disables the XATMI trace function (= trace function for the X/Open interface XATMI), see also openUTM manual “Creating Applications with X/Open Interfaces”.
	In UTM cluster applications, XATMI-TRACE applies locally to the node.
	For information on the names of the trace files, see “Trace files”.
ERROR	The XATMI trace function is enabled with the level ERROR at the start of the application. Only errors are logged.
INTERFACE	The XATMI trace function is enabled with the level INTERFACE at the start of the application. The level INTERFACE includes the level ERROR, and all XATMI calls are also logged.
FULL	The XATMI trace function is enabled with the level FULL at the start of the application. The FULL level includes the INTERFACE level. All KDCS calls to which the XATMI calls are mapped are also logged.
DEBUG	The XATMI trace function is enabled with the level DEBUG at the start of the application. The level DEBUG includes the level FULL, and diagnostic information is also logged.
OFF	The XATMI interface trace function remains disabled at the start of the application. Default: OFF

Trace files

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

KDC.TRC.trace-type.appliname.pid (standalone application) or

KDC.TRC.trace-type.appliname.nodename.pid (UTM cluster application)

trace-type

Identifies the trace type:

ADMI ADMI trace

CPIC CPI-C trace

TX TX trace

XATMI XATMI trace

appliname

Name of the application

nodename

Name of the node on which the node application is running

pid

PID of the process

Sample contents of a start parameter file of a standalone application

```
.UTM START FILEBASE=/home/utmsmp (Unix and Linux systems)
.UTM START FILEBASE=C:\utmtest\utmsmp (Windows systems)
.UTM START TASKS=2
.UTM START TASKS-IN-PGWT=1
.UTM START ASYNTASKS=1
.UTM START TESTMODE=OFF
.UTM START BTRACE=OFF
.UTM START ADMI-TRACE=ON
.UTM START OTRACE=OFF
.UTM START STXIT=ON
.UTM END
```

4.6 Cold start and warm start

These terms are explained below for openUTM:

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

Cold start with openUTM

Before an application starts for the first time, you have created the KDCFILE using the UTM generation tool KDCDEF. Following a regeneration of the KDCFILE or if a UTM application has been normally terminated first, openUTM performs a cold start the next time the application is started. Once it has started successfully, openUTM issues the following message:

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

Warm start with openUTM

If a UTM application has been abnormally terminated, openUTM performs a warm start the next time this application is started. During a warm start, openUTM brings the KDCFILE into a consistent state. Once it has started successfully, openUTM issues the following message:

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

You should note that UTM-S and UTM-F differ in the scope of their restart functions. See also chapter "Fault tolerance and the restart function" in the openUTM manual "Concepts und Functions".

If a UTM database application terminated abnormally (operating system crash or UTM application crash), the administrator of the database system must bring the database to a correct state before the warm start. When a warm start is carried out for a UTM database application, openUTM implements a common recovery phase.

4.7 Error messages at the application start

If the start of a UTM application or of a process is terminated due to an error, openUTM generally outputs message K049 and/or K078. Message K078 can occur in several variants. A detailed description of these messages and the contained return codes can be found in the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”.

Start errors can occur at the start of every work process.

5 Terminating a UTM application

A UTM application can be terminated as follows:

- normally via the administration or the KDCSHUT tool or
- abnormally as a result of a bottleneck of operation resources, as a result of internal errors in openUTM, or via the administration.

After an application terminates, you may still have to release global system resources before the application can be restarted. See also [section “The KDCREM tool”](#).

A number of special issues need to be taken into account when terminating a UTM cluster application. For information see the [section “Shutting down a UTM cluster application”](#).

5.1 Terminating a UTM application normally via administration

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

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

or

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

or

```
KDCSHUT NORMAL
```

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

Instead of the KDCSHUT command, you can also use the corresponding function in WinAdmin/WebAdmin or KDCADMI.

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

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

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

5.2 The KDCSHUT tool - terminating a UTM application normally at shell level

The KDCSHUT tool is a simple method of terminating the application without having to sign on to the application as the administrator. The KDCSHUT tool has the same effect as the administration command KDCSHUT N or KDCSUT G, TIME= or the command KDCSHUT W, TIME=.

The KDCSHUT tool is activated as follows:

Unix and Linux systems: `utmpath/ex/kdcshut filebase [time [G]]`

Windows systems: `utmpath\ex\kdcshut filebase [time [G]]`

filebase

is the base name of the application;

time

is the wait time in minutes until the application terminates.

Maximum wait time: 60 minutes

G

terminates the application with a graceful shutdown (see the openUTM manual “Administering Applications”).

5.3 Terminating a service on Windows systems

If a UTM application has been started as a service, then it can either be terminated as though it were not started as a service (i.e. for example with the KDCSHUT tool) or can be terminated as a service. You do this in a manner similar to that of the start procedure:

1. Sign on using a Windows user ID that has administration privileges.
2. Call the control panel with *Start - Control Panel*.
3. Click on *Administrative Tools* and then on *Services* and highlight the desired UTM service; this service is always named `openUTM servicename`.
4. Press the *Stop* button; the service and therefore the application are terminated normally.

If the Windows system is shut down, then the service and therefore the application are also terminated normally.

5.4 Terminating a UTM application abnormally

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

- internal UTM error
- error in the system environment and shutdown of the Unix or Linux system
- UTM administration command KDCSHUT KILL (or by the corresponding WinAdmin/WebAdmin or KDCADMI function)
- user error

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

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

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

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

When the error has been rectified, you can restart the application, and openUTM executes a warm start automatically.

- A system error has occurred. In this case, produce diagnostic documentation and write a problem report to the system support personnel. To do this, you must edit the UTM dumps of all work processes of the application using the `KDCDUMP` tool. Further documentation includes the *stdout* and *stderr* system files, the *gcores* (for Unix and Linux systems), the *utmwork* program, the KDCDEF control statements, and an evaluation of the system log file.

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

If the application terminates abnormally, the KDCREM tool must be called before restarting the application (see following section).

5.5 The KDCREM tool

The KDCREM tool is used to delete or reset any remaining semaphores and shared memories, as well as the status information relating to the application contained in the `applifile` file in the `utmpath` after the application ends. For more information, see [section “Global system resources of an application”](#).

! CAUTION!

- Following an abnormal termination of the main process of a UTM application (e.g. by a operating system error, system shutdown, or the SIGKILL signal), KDCREM **must** be called before the application is restarted.
- The KDCREM tool abnormally terminates a running UTM application without any warning!

KDCREM call

Unix and Linux systems: `utmpath/ex/kdcrem filebase`

Windows systems: `utmpath\ex\kdcrem filebase`

`filebase` is the base name of the application whose semaphores, shared memories and status information in the `applifile` file in `utmpath` are to be deleted.

6 UTM database application

This chapter provides a comprehensive overview of how to implement databases (= resource managers) under openUTM. The XA interface standardized by X/Open is used by openUTM for linking.

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

- Oracle



More details on the concept of coordinated interoperation can be found in the openUTM manual “Concepts und Functions”.

6.1 Generating a UTM database connection

You must generate the UTM database connection in the KDCDEF statement RMXA. Here you specify:

- the name of the *xa_switch_t* structure as preset by the database used
- for Windows systems: whether the *xa_switch_t* structure is addressed with *dllimport*; when linking with Oracle, *dllimport* must always be used for addressing
- Database access data (user name, password).

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

As a rule, there is a static and dynamic XA switch. A database can provide one or both variants. If the database provides a dynamic XA switch, you should use this, as this minimizes resource occupancy in the database system.

For further details on the RMXA statement, refer to the openUTM manual “Generating Applications” and the description of the RMXA statement.

Configuration of the dynamic XA switch of Oracle on Windows systems

When the dynamic XA switch *xaoswd* is used for the XA connection with Oracle on Windows systems, you must first extend the Windows Registry by adding the entry `ORA_XA_REG_DLL`.

To do this, perform the following steps after installing Oracle Client and openUTM:

1. Start the Windows Registry Editor in the Windows prompt with the `regedit` command.
2. Permit system access in the *User Accounts* tab of the control panel.
3. Under `HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE` select the key `KEY_OraClientnnHome1`, e.g. `KEY_OraClient12Home1`.
4. In the *Edit* menu select the command *New -> Expandable String Value* and enter the following:
 - For *Name* (default setting `New Value #1`) the value `ORA_XA_REG_DLL`
 - For *Type* the value `REG_EXPAND_SZ`.
 - For *Data* the fully qualified name of the installed dynamic openUTM library (*libwork.dll*), e.g. `C:\openUTM-Server\ex\libwork.dll`.
5. Confirm with OK and terminate the Registry Editor.

The Windows system is thus configured with openUTM and Oracle for use with the dynamic Oracle XA connection.

Important!

To ensure that the XA connection between openUTM and Oracle was configured correctly, the following must be checked once: *xa_reg* calls are logged in the following manner by openUTM when logging of the CA interface is performed and when updates are integrated into the Oracle database

```
kdcrtxa-<pid>: AX_CALL-1: ax_reg, XID.GTRID=<utm-gtrid-value>
```

If the openUTM XA debug information contains no entries of the aforementioned type, or if the Oracle XA debug information contains *xa_reg* calls which display the NULL XIDs, a serious configuration problem exists!

You enable the logging of the XA interface by means of the "XA debug" function (e.g. with `KDCDIAG`), see openUTM manual “Administering Applications” and [section “Debug parameters”](#).

6.2 Linking a UTM database application on Unix and Linux systems

For the UTM database link, you must incorporate additional modules into the UTM work process. These modules are listed below for the individual database systems. Please check the user guide for the respective database system to ensure the accuracy of the names of modules which make up this database system.

i You can simplify matters by using the sample application supplied with openUTM; see also "[Sample application on Unix and Linux systems](#)". This sample application provides a simple means of creating a UTM database application that contains all the necessary database libraries. This database application can be used as a template for your own application; for example, you can adapt the generated makefile, see [section "Linking with a makefile \(Unix and Linux systems\)"](#).

Connection to Oracle

A number of Oracle modules must additionally be linked in to set up a connection with Oracle. The sample programs and procedures supplied with Oracle indicate what modules are involved. The Oracle client library is named `$ORACLE_HOME/lib/libclntsh.so`. In case of ESQL-COBOL applications you also have to link the object module

`$ORACLE_HOME/precomp/lib/cobsqlintf.o`.

The list of system libraries additionally used by Oracle is located in the file `$ORACLE_HOME/lib/sysliblists`.

The preprocessor flag `release_cursor=yes` must be set in all cases. For information, see the Oracle User Guide.

6.3 Linking a UTM database application on Windows systems

On Windows systems, a UTM database application is created in the same way as a UTM application; see "[Creating application programs on Windows systems](#)". Only the following additional options need be set:

- For the Visual Studio options, you must specify the directory containing the database libraries, see "[Application programs in C and C++ \(Windows systems\)](#)".
- For the linker options, you must specify the name of the necessary database libraries, see "[Compiling and linking the application \(Windows systems\)](#)".

This ensures that the correct database libraries will be linked when the linker is called; see "[Compiling and linking the application \(Windows systems\)](#)".

i You can use the Quick Start Kit to create a UTM database application, see "[openUTM Quick Start Kit on Windows systems](#)". The Quick Start Kit is supplied with openUTM.

6.4 Starting and stopping a UTM database application

A UTM database application can be started and stopped in the same way as a UTM application, i.e. by starting and stopping the UTM application program.

6.4.1 Start parameters for a UTM database application

To start a UTM-DB application, you must specify the database start parameters in addition to the openUTM start parameters. The following schema applies here:

.UTM ...

Start parameters for openUTM, see the [section "Start parameter file of the application"](#).

...

.RMXA [xa-inst-name] ...

Start parameters for the database system (XA Resource Manager). These are described in the manual for the DB system. You can find examples in the Unix and Linux systems sample application, for Windows systems in the Quick Start Kit, and in the following sections.

END

Start parameters for the database system have the prefix ".RMXA". openUTM then forwards these start parameters to the Resource Manager when the application starts. The Resource Manager is opened by openUTM during the start phase of the UTM work process.

6.4.1.1 Openstring and Closestring

In the start parameter file, you define the database (instance of the Resource Manager) using an open string and, if required by the Resource Manager, you specify a close string. The database system Oracle do **not** require a close string.

The specifications for the open string and close string must be supplied by the respective Resource Manager. The syntax of these specifications therefore also depends on the particular Resource Manager and can be found in the manual for the Resource Manager used. openUTM transfers the strings from the start parameter file to the Resource Manager without checking them. Each string must be enclosed in double quotes and can be up to 255 characters long.

The open string and close string are specified in **one** line in the start parameter file, separated by a blank (a close string is only specified if required by the Resource Manager):

```
.RMXA RM="name" , OS="openstring" [CS="closestring" ]
```

The line can contain a maximum of 560 characters in total.

Specifying access data for the Oracle database

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

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

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

The access data can be changed later by administration.

6.4.1.2 Several instances

The UTM application can operate several entities (databases) of the Resource Manager, provided the Resource Manager supports multi-instance mode. In this case, you must specify a separate open string for each instance. Each open string must be entered in a separate line in the start parameter file. The name of the Resource Manager *name* must match the individual start parameter statements. For the open strings (databases), you must specify various names (entered in the DB= parameter within the strings, e.g. +DB=DBNAME1 and +DB=DBNAME2).

The general syntax for start parameters of several XA instances of a Resource Manager is:

```
.RMXA[ xa-inst-name1] RM= " name" , OS= " openstring1"  
.RMXAxa-inst-name2 RM= " name" , OS= " openstring2"  
.RMXAxa-inst-name3 RM= " name" , OS= " openstring3"  
...
```

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

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

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

If the Resource Manager requires a close string, a close string must also be specified for each instance.

! CAUTION!

In conjunction with a linked database connection (generated in openUTM), there must be no unlinked database connection.

For examples of start statements to the database system with which openUTM can be linked, refer to section [“Example of Oracle start parameters”](#).

Example

This example illustrates the connection of two Oracle XA databases to openUTM (Unix/Linux/Windows).

Start parameters:

```
.RMXA  
RM= "Oracle_XA" , OS= "Oracle_XA+Acc=P/*UTMUSER/*UTMPASS+SqlNet=RAC2+SesTm=60+LogDir=.  
+DbgFl=0"  
.RMXAEVE  
RM= "Oracle_XA" , OS= "Oracle_XA+DB=EVESEDB+Acc=P/*UTMUSER  
/*UTMPASS+SqlNet=RAC1+SesTm=60+LogDir=. +DbgFl=0"
```

The first start parameter contains only the prefix ".RMXA", i.e. an empty XA instance name, and is therefore assigned to the first XA Resource Manager generated in openUTM without an XA-INST-NAME parameter. In other words the first XA instance sets up a connection with the correspondingly generated access data ("MaxTheSuperman", etc.).

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

KDCDEF generation:

```
RMXA XASWITCH=xaoswd,USERID='MaxTheSuperman',PASSWORD='PasswordOfMax'  
RMXA XASWITCH=xaoswd,USERID='Eve',PASSWORD='PasswordOfEve',XA-INST-NAME=EVE
```

6.4.1.3 Example of Oracle start parameters

Oracle only requires an open string, no close string.

Multi-instance mode is possible, i.e. several open strings can be specified for the Resource Manager in the start parameter file of a UTM application.

The following start parameters, for example, can be specified in the start parameter file for an Oracle database:

```
.RMXA Oracle_XA OS="Oracle_XA+Acc=P//+SesTm=60"
```

This statement must be written in one line without a line feed.

How you obtain DEBUG information for the connection to the database is described in [section "Debug parameters"](#).

Only the mandatory parameters are listed in the open string. In addition, you can specify other optional parameters. These are listed further below.

The parameters in the open string are separated by the "+" character.

Meaning of the mandatory parameters:

Oracle_XA	Name of the Resource Manager prescribed by Oracle, as contained in the <i>xa_switch</i> structure.
Acc=P//	Information for data access control to the database (user access information). If <i>Acc=P//</i> is specified as in the example, neither a user ID nor a password are transferred for data access control.
	An Oracle database can also request database-specific information (<i>user</i> and <i>pwd</i>) which must be transferred with <i>Acc=P/user/pwd</i> . For further information, see the Oracle manual.
SesTm=	Maximum time in seconds available for a transaction (60 s in the example). Possible specifications for <i>SesTm</i> can be found in the Oracle manual.
	Specifying <i>SesTm=0</i> means that there is no restriction on the duration of the transaction. It is therefore recommended to specify a value > 0 for <i>SesTm</i> .

The parameters listed below are optional. Please refer to the Oracle manual for an explanation and description of possible definitions.

DB=	Name of the Oracle database.
	You must specify this parameter if the UTM application is to be linked with more than one Oracle database (multi-instance mode).
GPwd=P/	Group password; specified in the form <i>GPwd=P/password</i> .
LogDir=	Path name of the logging directory.
MaxCur=	Maximum number of open cursors.
SqlNet=	Network connection string.

i No action is required if the application is terminated abnormally, because openUTM automatically carries out a common recovery phase before the UTM application is restarted.
Please note that, on Windows systems, the RMXA statement needs the DLLIMPORT=YES operand when generating with KDCDEF.

Information on the objects/libraries to be linked and on the start parameters (openstring) can be found in the Oracle documentation.

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

The access authorization for an Oracle database should be defined via KDCDEF generation for security reasons.

Please note the following:

- The Oracle user name for the connection to Oracle and the associated Oracle password must be generated in KDCDEF (KDCDEF statement RMXA, USERID and PASSWORD operands).
The Oracle password is stored as a hashcode in the UTM system tables (masked) and is therefore not present in clear text in the UTM dump.
- In the openstring for the start parameter, specify the placeholder *UTMUSER in place of the Oracle user name and the placeholder *UTMPASS instead of the Oracle password. These placeholders are replaced in accordance with the following rules:
 - If the openstring contains at least one of the placeholders *UTMUSER or *UTMPASS, then UTM replaces the placeholders with the values generated for the specific database system on an xa_open() call. I.e. in the open string, *UTMUSER is replaced by the generated Oracle user name and *UTMPASS by the generated Oracle password.
For security reasons, the Oracle password is converted into clear text only immediately prior to use on an xa_open() call and is then deleted in the process memory immediately after the xa_open() call.
 - If the openstring of the start parameter does not contain either *UTMUSER or *UTMPASS then it is passed unchanged to the xa_open() call.

Please note that processing is case-sensitive!

Example

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

```
OS="Oracle_XA+SqlNet=011+ACC=P/*UTMUSER/*UTMPASS+DbgFl=15"
```

Behavior if the Oracle access data is not generated

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

6.4.2 Start parameters for failover with Oracle® Real Application Clusters

A UTM application communicates with Oracle Real Application Clusters over the XA interface. If an XA call cannot be executed correctly by Oracle in the event of a failover, Oracle returns the value "XAER_RMFAIL".

In normal circumstances, i.e. when failover support is not activated, openUTM takes this message to mean that it is no longer possible to work with this database and aborts execution of the application.

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

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

RAC=Y

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

RAC=N

Disables failover support.

Default: N

RAC_retry=nnn

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

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

Default: 1

RAC_recover_down=

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

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

Default: N

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

Behavior of openUTM in the event of failover

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

- The application is not aborted if failover to a node of the Oracle® Real Application Cluster is possible.

-
- If the connection is lost between "Prepare" and "Commit" at the end of a transaction, a "Reconnect" with recovery is performed and if this is successful, the "Commit" operation is repeated over this new connection.
 - If transactions are still open when the failover occurs, this can still lead to problems and corresponding error messages even if failover support is enabled (e.g. return code ORA-25402 - transaction must roll back). The reason for this is that Oracle® Real Application Clusters is unable to migrate any open transactions in the event of a failover. These transactions must be rolled back by the UTM application program, see also ["Interrupted transactions"](#). Any open multi-step transactions (i.e. following PEND KP) are rolled back by the database system in the event of a failover. openUTM has no influence over this. The database system is automatically reconnected after the rollback. It is then possible to start new transactions.
 - If the failover occurs during a warm start of the application or while the UTM process is being terminated, error processing is carried out as usual and no attempt is made to reconnect.
 - The "prepared statements" database function can lead to errors in the event of a failover.
 - Messages allow the progress of the reconnection to the database system to be monitored.
 - xa_close in the event of reconnection:
In &RMSTAT insert in message K202, the string "RAC closed" is output for the Oracle® Real Application Clusters instance in place of "closed". Oracle® Real Application Clusters instance in place of "closed".
 - xa_open in the event of reconnection:
In the &XACALL insert of message K224, the string "RAC: xa_open" is output.

Debug messages

The debug messages contain an indication whether the message refers to an instance of Oracle® Real Application Clusters.

How you obtain XA DEBUG information for the connection to the database is described in [section "Debug parameters"](#).

Interrupted transactions

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

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

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

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

- One transaction is interrupted by the failover; at this time, the UTM process is still connected to the old node.
- After the process is restarted or after a PEND ER in the UTM application program, the interrupted transaction is continued by a different UTM process. This process is now connected to the new node.

-
- The database instance rejects the request to resume the interrupted transaction (xastart with RESUME) and reports that the transaction is unknown.
 - openUTM reconnects to the database instance. openUTM attempts to resume the transaction over the new connection (i.e. with the new node).
 - The database system again rejects this request, since the database transaction was started on the old node of the Oracle[®] Real Application Cluster and cannot be continued on the new node.
 - openUTM rolls back the global transaction and issues a K160 message; "NOTA" is output in the insert of the internal return code KCRCDC.

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

Control using a MSGTAC program

The MSGTAC event service is defined as the message destination for the K160 message.

MSGTAC reacts to the message insert and initiates a restart over the administration programming interface (KC_CHANGE_APPLICATION). This replaces all processes, restarts them and then connects them to the new node.

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

6.4.2.1 Special issues when connecting to Oracle®

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

This offers the following options for live operation:

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

Creating a DTP service (Oracle®)

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

Example:

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

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

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

and

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

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

2. Convert the services to "DTP services" using SQLPLUS:

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

You can start, stop and administer the (DTP) services with "srvctl commands". See also the Oracle® "Administration and Deployment Guide".

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

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

Example

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

4. In the Openstring in the start parameters, assign this `net_service_name` (in this case `RACUTMS1`) to the operand "SqlNet".

6.4.3 Debug parameters

You have the option of logging the XA interface in openUTM for test purposes. The RMXA start parameter `DEBUG=` is available for this purpose.

The `DEBUG=` parameter has the following format and should be used as the first RMXA start parameter:

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

Explanation

`DEBUG=`

Activates the debug function.

`YES`

Logs the individual XA calls and, for each call,

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

`ALL`

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

`OUTPUT=`

Specifies the output destination.

`SYSOUT`

Output is sent to *stderr*.

`FILE`

Output is sent to a file.

The file has one of the following formats:

`KDC.TRC.XA.appliname.pid` (standalone application)

`KDC.TRC.XA.appliname.nodename.pid` (UTM cluster application)

`appliname`

Name of the application

`nodename`

Name of the node on which the node application is running

`pid`

PID of the process.

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

6.4.4 Normal termination of a UTM database application

A UTM database application is terminated using UTM administration functions, see the [section “Terminating a UTM application normally via administration”](#). openUTM closes the Resource Manager while the UTM work process of the application is terminating.

If an application has been started as a service on Windows systems, it can be stopped as a service (see the [section “Terminating a service on Windows systems”](#)) or using the KDCSHUT utility (see [section “The KDCSHUT tool – terminating a UTM application normally at shell level”](#)).

6.4.5 Abnormal termination of a UTM database application

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

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

6.5 Operating a UTM database application

The operation of a UTM database application is based on the same principles as the operation of a UTM application. The special points to observe are described in the following sections.

6.5.1 User sign-on and sign-off

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

When signing on, users can avail themselves of all the sign-on options offered by UTM. In particular, the user can use the SIGNON services of UTM. The following must be noted here:

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



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

6.5.2 Diagnostics

To diagnose errors in a UTM database application, UTM offers the same information sources as for a pure UTM application, i.e. UTM messages, error codes, and dumps. Some of these sources also contain database-specific data, which should be examined first if an error could relate to a fault in the database connection. The following UTM diagnostic information is supplied:

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



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

7 UTM cluster application

A cluster is a number of computers (nodes) connected over a fast network and which share common peripherals.

A UTM application can run as a UTM cluster application on a cluster. To a large extent, a UTM cluster application can be operated as a single UTM application (standalone application). A UTM cluster application is made up of several identically generated UTM applications (the node applications) that run on the individual nodes.

On Unix, Linux and Windows systems, a UTM cluster application can run on up to 32 nodes.

7.1 Properties of a UTM cluster application

A UTM cluster application is intended to run on more than one computer. It has the following characteristics:

- The UTM cluster application has to run under the same user ID on all computers to ensure that the same access permissions apply to the files used.
- The configuration of the UTM cluster application, including the KDCFILE for all nodes, is created in a single generation run and is therefore the same for all nodes. This also applies in particular to the application name of the UTM cluster application.
- The computers that belong to a cluster must be compatible in terms of hardware status and software configuration. Discrepancies involving compatible correction statuses, operating system versions and updates are possible. For details, see the Release Note.
- The node applications of a UTM cluster application must all run under the same operating system (e.g. Solaris) with the same bit mode (32-bit or 64-bit on all nodes). Mixed configurations, such as Unix and BS2000 computers or even Solaris and Linux computers in combination are not possible.
- A number of files that can be accessed jointly by all nodes are required in order to run a UTM cluster application. These are the UTM cluster files. For detailed information on the UTM cluster files, refer to the [section “Runtime environment”](#).
- There are also files which are local to each node. A node application’s KDCFILE must be accessible from all node applications. You have to create these files with a node-specific filename prefix. For detailed information on the files local to the nodes, refer to the [section “Runtime environment”](#).

Special properties of a cluster on Unix and Linux systems

- To run a cluster application, the user IDs used must not only have the same names on all the computers, but must also be managed internally by the operating system using the same user number assigned when the user ID was set up.
- In order to run scripts across different computers, the nodes must mutually permit ssh access for the execution IDs.
- On Unix and Linux systems, the **Network File System/Service (NFS4)** is used to access the common files. You can, for instance, use NetApp FAS as the NFS4 server system.

Special properties of a cluster on Windows systems

- The Windows computers must all be members of a common Windows domain.
- An identical Windows domain login must be used as the execution ID on all nodes.
- On Windows, Windows shares are used with the CIFS protocol common on Windows.

7.2 Installing and preparing a UTM cluster application for use

- Installation
 - Installing the UTM runtime components for Unix and Linux systems
 - Installing further runtime components for Unix and Linux systems
- UTM generation
 - Special UTM generation statements for UTM cluster applications
 - Generating reserve nodes
- Using global memory areas
- Service restart
- Runtime environment
 - Files
 - Location of the files
- Preparation for use
- Example for Unix and Linux systems

7.2.1 Installation

Before you can create and operate a UTM cluster application, you must install the product openUTM on all computers to be used for the cluster. The procedure used to install openUTM does not depend on whether you subsequently wish to operate standalone or UTM cluster applications. See also the sections [“Installing openUTM on Unix and Linux systems”](#) and [“Installing openUTM on Windows systems”](#).

You will find information on the software requirements for UTM cluster applications in the Release Note.

The runtime environment of openUTM (e.g. the system time) must be the same on all nodes. See [section “Properties of a UTM cluster application”](#).

A number of files that can be accessed jointly by all node applications are required in order to run a UTM cluster application. See [section “Runtime environment”](#).

openUTM revision levels can be deployed during live operation of a UTM cluster application. For details, refer to the [section “Use of openUTM revision levels in the UTM cluster application”](#).

If the applications on Windows systems are to run as services, you must install and configure the respective service on all nodes.

7.2.1.1 Installing the UTM runtime components for Unix and Linux systems

Since the applications are identified via their application names in the `applifile`, each node must use its own installation files, i.e. openUTM must be installed at every node.

- > The UTM installation procedure proposes the installation directory `/opt/lib/<utmversion>`, e.g. `/opt/lib/utm70a00`.
- > If installation under `/opt/lib/<utmversion>` is not possible then we urgently recommend that you choose a directory with the same name on all the nodes when performing openUTM installation and that each of these directories is located on a hard disk that is exclusively assigned to the respective node.
- > If this type of uniform installation is also not possible then please note the following points:
 1. When linking the application using the binder `ld`:
 - > Specify the directories containing the shared objects by means of the flag `-L`, e.g. `-L/opt/lib/utm70a00/64/sys`.
 - > Specify the names of the shared objects using the flag `-l` without the prefix `lib` and without a suffix, e.g. `-lwork` in order to link `libwork.so`.
 - > Respect the sequence of these flags: Specify the flag `-L` before the flag `-l`.
Rule: "Capital L before small l" due to the risk of confusion with "i" when using uppercase.
 2. At the start of the application set the environment variable `$LD_LIBRARY_PATH` and, if necessary, `$LD_LIBRARY_PATH64` to indicate the directories containing the employed shared objects.

7.2.1.2 Installing further runtime components for Unix and Linux systems

- > Install any other runtime components that are used by the UTM application (e.g. Cobol runtime system or database software) as uniformly as possible on all the nodes, i.e. in the same directories.

This ensures that access to these runtime components is possible consistently and via the same paths (e.g. including during node recovery) from each node.

- > If this type of uniform installation is not possible then respect the notes on linking and starting presented in the description in [section "Installing the UTM runtime components for Unix and Linux systems"](#).

7.2.2 UTM generation

Configuration of the UTM cluster application including the initial KDCFILE is created in a common generation run.

You create the initial KDCFILE for a UTM cluster application in the basic generation run. It is stored under the base name that you specify in the KDCFILE operand of the MAX statement.

7.2.2.1 Special UTM generation statements for UTM cluster applications

Special UTM generation statements are required for generating a UTM cluster application:

- The CLUSTER statement defines the common properties of the UTM cluster application.
- The CLUSTER-NODE statements define the computers on which the node applications will run and specify the node-specific properties for each node application. You must issue a separate CLUSTER-NODE statement for each node application.

i The number of CLUSTER-NODE statements specifies the number of node applications for the cluster. You cannot subsequently add further node applications to the cluster in live operation. You can, however, create "reserve" nodes during UTM generation and subsequently modify these using the administration facilities, populating them with actual values for additional nodes. See below.

openUTM manual “Generating Applications”



CLUSTER statement

The CLUSTER-FILEBASE operand specifies the name prefix that is global to the cluster for the files of the UTM cluster application that are global to the cluster.

CLUSTER-NODE statement

The FILEBASE operand specifies the base name for the node application that is local to the node.

7.2.2.2 Generating reserve nodes

During generation with KDCDEF, you have the option of creating reserve nodes 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.

This option is particularly useful in the following situations:

- You generate more nodes than you initially wish to operate as a reserve, for instance because insufficient computers are yet available.
At a subsequent time, you wish to add a node to an existing cluster because the number of nodes that were available to date is no longer sufficient. Now that you know the data of the new node, you can use the administration facilities to modify the configuration of a reserve node.
- The hardware on which a node application is running is faulty or is to be replaced by more powerful hardware. To do this, proceed as follows:
 - Terminate the node application.
 - Transfer the UTM application data to the new computer.
 - Use the administration facilities in a running node application to change the computer name of the terminated node, i.e. enter the new name of the node instead of the old computer name here.

After you have made the change, you can start the node application on the new computer.



You will find detailed information on generating reserve nodes and on modifying the provisional properties using the administration facilities in the openUTM manual “Generating Applications” and the openUTM manual “Administering Applications”.

7.2.3 Using global memory areas

In UTM cluster applications, the UTM storage areas GSSB and ULS are supported at the global cluster level. The associated user data is stored in the cluster page pool.



openUTM manual “Generating Applications”, CLUSTER statement

You use the operands PGPOOL and PGPOOLFS to define the properties of the cluster page pool (size, warning level and number of files). You use the DEADLOCK-PREVENTION operand to control how the system behaves in the case of locked, global storage areas (additional check or control via timeout).

TACs for accessing GSSB and ULS

In UTM cluster applications, you should assign TAC classes to programs that access GSSB or ULS storage areas. By restricting the processes working for these TAC classes, you can prevent all the processes in a node application from simultaneously accessing the GSSB or ULS storage areas. UTM rejects attempts to access storage areas if this would mean that all the processes in a node application would have to wait for a lock held by another node.

If it is possible, it is advisable to place the TACs that access GSSB or ULS in the same TAC class. Any TACs that use PGWT should be gathered together in the same TAC class since it is also necessary to take account of the PGWT wait situations.

When you have assigned the TACs to TAC classes, you can restrict the number of processes by means of either the TACCLASS or TAC-PRIORITIES statement:

- TACCLASS statement:
The number of processes that are started must be at least one greater than the maximum number of processes that are allowed to run for the TAC classes containing the TACs which access GSSB or ULS.
- TAC-PRIORITIES statement:
The number of processes that are started must be at least one greater than the total of FREE-DIAL-TASKS and MAX ASYNTASKS.

Examples

In the example below, TASKS=10 and ASYNTASKS=2 are generated in the MAX statement. The TACs with GSSB /ULS access are to run in TAC class 2 (TAC... TACCLASS=2). This means:

- If the process limitation is controlled via the TACCLASS statement and TAC class 2 is able to use a maximum of 5 processes then the TACCLASS statement is as follows:

```
TACCLASS 2 , TASKS=5 , PGWT=YES
```

At least 6 processes must be started.

- If the process limitation is controlled via the TAC-PRIORITIES statement and at least one process is to be kept free for jobs whose TACs do not belong to any dialog TAC class then the TAC-PRIORITIES statement is as follows:

```
TAC-PRIORITIES FREE-DIAL-TASKS=1
```

At least 4 processes must be started (because MAX ... ASYNTASKS=2).

7.2.4 Service restart

In UTM cluster applications, service restarts are supported globally throughout the cluster for all genuine user IDs generated with RESTART=YES. This means that after signing off at the node application, a user is able to continue an open dialog service at another node application provided that the service is not a node-bound service.

Node-bound services

The following services are node-bound:

- Services that have started communication with a job receiver via LU6.1 or OSI TP and the job receiver service has not yet been terminated
- Inserted services in a service stack

In addition, a service associated with a user is node-bound as long as the user is signed on at a node application. Hence, 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.

Node-bound services can only be continued at the node to which they are bound.

If a user who has a node-bound service wants to sign on at another 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, or
- the UTM cluster application has been generated with ABORT-BOUND-SERVICE = NO.

If an attempt by a user with a node-bound service to sign on at another node application is accepted, then the open service is not continued but is instead terminated abnormally the next time the node application to which it is bound is started.



- A connection user ID is bound to the connection. A connection user ID generated with RESTART=YES can have an open service in every node application.
- In applications without USER, an LTERM generated with RESTART=YES can have an open service in every node application.

Service restarts in UTM-F applications

Although service restarts are also supported in UTM-F applications, the service data is not saved until the user signs off.

As a result, following an abnormal termination of a node application, no further service restart is possible if the user

- was signed on at the node application at the time it was terminated abnormally or
- has a service bound to the node application that has terminated abnormally.

7.2.5 Runtime environment

- [Files](#)
- [Location of the files](#)

7.2.5.1 Files

Both files that are global to the cluster and files that are local to the node belong to the runtime environment of the UTM cluster application.

You specify in the storage location of the files during UTM generation using the following KDCDEF statements:

- `CLUSTER CLUSTER-FILEBASE = cluster_filebase`
cluster_filebase identifies the storage location of the UTM cluster files.
- `CLUSTER-NODE FILEBASE = node_filebase`
node_filebase identifies the storage location of the files local to the node.

You must specify *cluster_filebase* for the application run when the node applications are started using the start parameter `CLUSTER-FILEBASE = cluster_filebase`. The same value must be specified for this start parameter for all node applications.

i The value that you specified for *cluster_filebase* during UTM generation does not have to match the value that you specify for *cluster_filebase* using the start parameter.

It is crucial that the UTM cluster files, such as the cluster configuration file, are available under the base name (*cluster_filebase*) specified in the start parameters at the time at which the first node application is started.

UTM cluster files

A number of files that can be accessed jointly by all node applications are required in order to run a UTM cluster application. These UTM cluster files are created in a base directory specific to the UTM cluster application (*cluster_filebase*).

The following list indicates all the UTM cluster files. In this list, the file names are specified without base directory. The complete name in each case is as follows:

cluster_filebase/UTM-C.xxxx on Unix and Linux systems

cluster_filebase\UTM-C.xxxx on Windows systems

xxxx=CFG, USER, ..., LOCK

UTM-C.CFG *)	Cluster configuration file Contains the configuration of the cluster, the current status of all the nodes of the cluster, additional information on all the node applications of the UTM cluster application and specifications on data that is global to the cluster.
--------------	---

UTM-.USER *)	<p>Cluster user file Contains user-specific information for managing users in a UTM cluster application.</p> <div style="border: 1px solid #add8e6; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>i In a UTM cluster application without explicitly generated user IDs, the cluster user file is not needed and is therefore not generated.</p> </div>
UTM-.CP nn *) (nn = 01, ..., 10)	<p>Cluster page pool files, the number of which is defined during UTM generation. Contain user data that is managed globally throughout the cluster in UTM cluster applications (GSSB, ULS and the service data of users).</p>
UTM-C.CPMD *)	<p>Control file for the cluster page pool</p>
UTM-C.GSSB *)	<p>Cluster GSSB file Used for GSSB management in a UTM cluster application</p>
UTM-C.ULS *)	<p>Cluster ULS file Used for ULS management in a UTM cluster application.</p>
UTM-C.JRN1 UTM-C.JRN2	<p>Administration journal which logs global administration actions ("memory" for the administration functions, see section "Administration journal"). openUTM uses these files to ensure that global administrative changes apply globally and consistently across the entire cluster.</p>
UTM-C.JKAA	<p>Journal file containing a copy of the KDCS Application Area (KAA). Administrative changes which are no longer contained in the administration journal (see section "Administration journal") are taken from this file.</p>
UTM-C.LOCK	<p>Cluster lock file Used for the management of queues in a UTM cluster application.</p>

UTM-C.SLCK	Lock file for serialization of the start phase of the node applications.
------------	--

The UTM cluster files indicated by *) are created by KDCDEF at generation time (see [section “UTM generation”](#)).

The journal files (.JRN1, .JRN2, .JKAA) and the lock files are set up by openUTM the first time the first node application is started.

! CAUTION!

You must not rename any of these files or copy them to a different location. This applies during operation of the UTM cluster application and after the UTM cluster application has been terminated.

Files local to the node

Both files that are global to the cluster and files that are local to the node belong to the runtime environment of the UTM cluster application. A filename prefix that is unique within the cluster (*node_filebase*) is assigned to each node in the case of files that are local to the node. There are the following files local to the node for each node application:

- the KDCFILE files (including the pagepool and restart areas) in the form of copies of the initial KDCFILE files:

node_filebase /KDCA

node_filebase \KDCA

node_filebase /PxxA

node_filebase \PxxA

if required by the UTM generation

node_filebase /RxxA

node_filebase \RxxA

if required by the UTM generation

The initial KDCFILE files are created using KDCDEF (see [section “UTM generation”](#)). You must copy these files for each node application.

You must organize the KDCFILES of the node applications in such a way that all KDCFILES of the node applications can be accessed by all other node applications.

- System log file (SYSLOG file)

node_filebase /SYSLOG

node_filebase \SYSLOG

The system log file SYSLOG can be a single file or a file generation group (FGG).

- User log file

node_filebase /USLA

node_filebase \USLA

The user log file USLOG must be a file generation group (FGG).

- Execution logs

- Diagnostics files

- Other application-specific files

You must set up the SYSLOG file and the user log file and other application-specific files for each node application.

i You can maintain different versions of the application program on Unix, Linux and Windows systems. The same version of the application program must, however, be loaded on all running node applications with a KDCFILE from the same generation run.

7.2.5.2 Location of the files

On Unix and Linux systems

- The cluster filebase must be located on a file system that can be accessed from all nodes. This means that it will typically be located on a storage subsystem that can be accessed with NFS4.
- For reasons of consistency, directories with the same names must be used as NFS4 mount points on all nodes.
- It is recommended that a common mount point is used for the cluster filebase and all filebase directories.

On Windows systems

- The cluster filebase must be on a network share that can be accessed from all nodes.
- On Windows systems, network shares are used that are accessed using the CIFS protocol.

Windows network shares can be addressed using the following two formats:

- Using drive letters, e.g. C:\
- Using the UNC name, i.e. in the format `\\ServerName\ShareName\`

These drives must be able to be accessed from all nodes using the same format. You must specify this name accordingly during UTM generation.

7.2.6 Preparation for use

Distributing the KDCFILE

In order to be able to run, every node application requires a copy of the initial KDCFILE from the common UTM generation run with a base name assigned exclusively to this node application. To achieve this, you must copy the initial KDCFILE (including the pagepool, restart areas are maintained) into the associated node-specific filebase for each node application after the UTM generation run.

Creating the start parameter file

When you start a node application, you must specify the start parameter CLUSTER-FILEBASE in place of FILEBASE. See also the [section “Start parameters for openUTM”](#). The files that are global to the cluster must be present under the base name specified in CLUSTER-FILEBASE.

i If you wish to specify a different name for *cluster_filebase* in the start parameter file than you set in the KDCDEF statements, you must rename the UTM cluster files generated by KDCDEF before the first node application is started.

7.2.7 Example for Unix and Linux systems

A common mount point is used for the cluster filebase and the filebase directories of all node applications in this example:

The volume `/vol/vol1` of an NFS4 system (e.g. NetApp Filer) that can be addressed in the network under the name `MyFiler` is mounted in the local directory `/myVol1` on all nodes.

This requires the following actions to be performed on all nodes:

login root

```
mkdir /myVol1 (only required before the first mount command)
```

```
mount -t nfs4 MyFiler:/vol/vol1 /myVol1 (for Linux)
```

```
mount MyFiler:/vol/vol1 /myVol1 (for Solaris)
```

i Note that the `mount` command must be repeated every time the computer is rebooted. It is recommended that you ask the system administrator to automate the mounting operation.

Directories

In this example

- two nodes are generated: `UTMHOST1`, `UTMHOST2`
- two node applications of the UTM cluster application use the common mount point `/myVol1`
- the directory `/myVol1/UTMCAPPL` is used as the cluster filebase
- the directories for the node applications (node filebase) are located below the cluster filebase and are designated according to the relevant host name. This directory structure improves clarity and is largely self-explanatory.

<code>/myVol1/</code>	Mount point
<code>/myVol1/UTMCAPPL/</code>	Cluster filebase
<code>/myVol1/UTMCAPPL/UTMHOST1/</code>	Filebase for UTMHOST1
<code>/myVol1/UTMCAPPL/UTMHOST2/</code>	Filebase for UTMHOST2

Both node applications must have access permissions on the cluster filebase and on the node-specific directories.

KDCDEF statements

The following UTM generation statements are required to generate the UTM cluster application in this example:

```

OPTION GEN=(KDCFILE,ROOTSRC,CLUSTER)

CLUSTER CLUSTER-FILEBASE=/myVol1/UTMCAPPL,
        LISTENER-PORT=1234,BCAMAPPL=NAMECLT,
        CHECK-ALIVE-TIMER-SEC=60,USER-FILEBASE=/myVol1/UTMCAPPL

CLUSTER-NODE FILEBASE=/myVol1/UTMCAPPL/UTMHOST1,HOSTNAME=UTMHOST1
CLUSTER-NODE FILEBASE=/myVol1/UTMCAPPL/UTMHOST2,HOSTNAME=UTMHOST2
...

```

Storing the files

- Files that are global to the cluster

During the UTM generation run, the cluster configuration file (`UTM-C.CFG`) and a number of central cluster files are created in the directory `/myVol1/UTMCAPPL`. The rest of the central cluster files are created when the first node application is started and also stored in the directory `/myVol1/UTMCAPPL`.

<code>/myVol1/</code>	NFS4 mount point
<code>/myVol1/UTMCAPPL/</code>	Cluster filebase
<code>/myVol1/UTMCAPPL/UTM-C.CFG</code>	Cluster configuration file
<code>/myVol1/UTMCAPPL/UTM-C.USER</code>	Cluster user file
<code>/myVol1/UTMCAPPL/UTM-C.CPMD</code>	Cluster page pool control file
<code>/myVol1/UTMCAPPL/UTM-C.CP01</code>	Cluster page pool file
<code>/myVol1/UTMCAPPL/UTM-C.GSSB</code>	Cluster GSSB file
<code>/myVol1/UTMCAPPL/UTM-C.ULS</code>	Cluster ULS file
These files are created by KDCDEF during UTM generation.	
<code>/myVol1/UTMCAPPL/UTM-C.JRN1</code> <code>/myVol1/UTMCAPPL/UTM-C.JRN2</code> <code>/myVol1/UTMCAPPL/UTM-C.JKAA</code>	Administration journal created the first time a node application is started.
<code>/myVol1/UTMCAPPL/UTM-C.LOCK</code>	Cluster lock file
<code>/myVol1/UTMCAPPL/UTM-C.SLCK</code>	File for serializing the start up of individual node applications
These files are created when a node application is started for the first time.	

- Initial KDCFILE:
 - You must copy the initial KDCFILE created during this UTM generation run into all filebase directories of the node applications.

- Files in the filebase directory on the node UTMHOST1

/myVol11/UTMCAPPL/UTMHOST1/KDCA	KDCFILE for UTMHOST1 Must be copied before the node is started for the first time.
/myVol11/UTMCAPPL/UTMHOST1 /utmwork	Program for the <code>utmwork</code> process of UTMHOST1. You must make this available before the first time the node is started.
...	Other files in the filebase directory on UTMHOST1

- Files in the filebase directory on the node UTMHOST2

/myVol11/UTMCAPPL/UTMHOST2/KDCA	KDCFILE for UTMHOST2 Must be copied before the node is started for the first time.
/myVol11/UTMCAPPL/UTMHOST2 /utmwork	Program for the <code>utmwork</code> process of UTMHOST2. You must make this available before the first time the node is started.
...	Other files in the filebase directory on UTMHOST2

Start parameter files

The cluster filebase must be specified in both start parameter files for the node applications:

```
...
.UTM START CLUSTER-FILEBASE=/myVol11/UTMCAPPL
...
```

i Note that the statement necessary for standalone applications `.UTM START FILEBASE=<filebase>` must not be contained in a start parameter file for a UTM cluster application. See also the [section "Start parameters for openUTM"](#).

7.3 Configuration of a UTM cluster application with a database

Because all node applications have an identical configuration, all node applications work with the same database system.

Using Oracle® Real Application Clusters (Oracle® RAC)

The following configuration is recommended if you are using Oracle® RAC:

One primary RAC node is assigned to each node application. In addition, each node application uses the other RAC nodes as fallback levels for failover purposes.

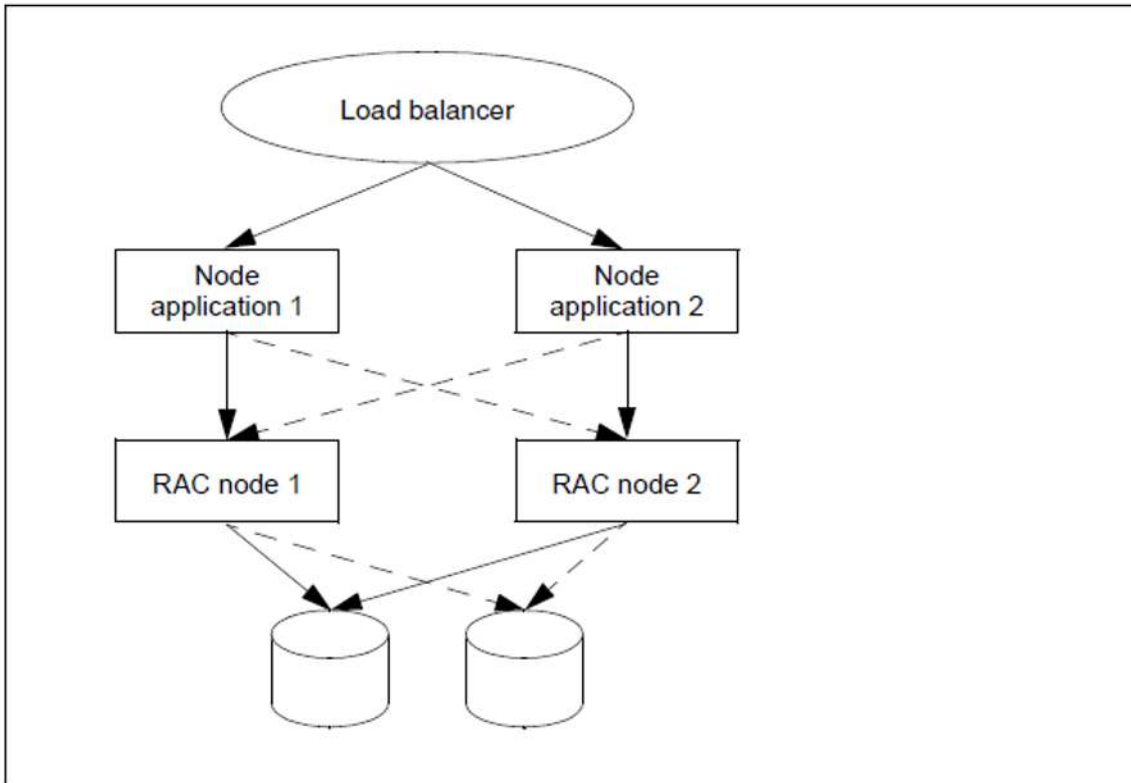


Figure 2: Configuration with two node applications and two Oracle® RAC nodes

7.4 Starting a UTM cluster application

i On Unix and Linux systems: Before starting the application, set the environment variable `LD_LIBRARY_PATH` and, if necessary, `LD_LIBRARY_PATH64` to indicate the directories containing the employed shared objects if you have not been able to use uniform installation paths (see also "[Installing the UTM runtime components for Unix and Linux systems](#)").

You start a UTM cluster application by starting one or more node applications. You start each node application separately in the same way as a standalone application (see the [section "Starting a UTM application on Unix and Linux systems"](#) and the [section "Starting a UTM application on Windows systems"](#)).

Start parameter file

In contrast to a standalone UTM application, the start parameter file must contain the statement `START CLUSTER-FILEBASE=cluster_filebase` in place of the statements `START FILEBASE=filebase`.

The following start parameters apply globally to the cluster:

- TESTMODE
- BTRACE
- OTRACE
- DUMP-MESSAGE
- *interval* value for SYSPROT

Start parameters which apply globally to all nodes are distributed from the first node application started to nodes which start subsequently via the administration journal. They remain valid – even during or after an update generation – until the UTM cluster application is terminated or until the value is changed using the administration functions.

If the node applications do not require any special start parameters, the start parameter file can be the same for all node applications. The UTM cluster files generated by KDCDEF must be present under the base name which you specified for CLUSTER-FILEBASE. These files must come from the same generation run (see the [section "Creating the startparameter file" \(Preparation for use\)](#)). The files of the KDCFILE must not be older than the UTM cluster files.

When a node application is started, the following cluster-specific start actions are performed:

- A check is performed whether the KDCFILE of the node application is compatible with the cluster configuration file.
- The first time the first node application is started, the administration journal files are initialized and the cluster lock file and the serialization file (UTM-C.SLCK) are set up.
- Cluster monitoring in which the node applications monitor each other is started when a second node application is started.
- Cluster monitoring is automatically extended when a further node application is started.
- The monitoring relationships are determined dynamically (see the [section "Application monitoring of the node applications"](#)).

SYSLOG file and user log file

You must set up the system log file SYSLOG and the user log file for each node application (see the sections "[System log file SYSLOG](#)" and "[User log file](#)").

The system log file SYSLOG must either be set up as a single file on all nodes or must be set up as a File Generation Group (FGG) on all nodes (see [“System log file SYSLOG”](#)).

All running node applications with a KDCFILE from the same UTM generation run must have the same SYSLOG configuration, otherwise startup of a subsequent node is aborted.

Encryption capability

You must ensure that the appropriate openssl library is provided on all nodes.

7.5 Monitoring of node applications and failure detection

Monitoring of node applications comprises

- an application monitoring
- and measures to be taken if a failure is detected, for instance starting a failure script.

7.5.1 Application monitoring of the node applications

If more than one node application has been started for a UTM cluster application, each node application is monitored by a different node application.

The following are dynamically defined when a node application is started:

- what other node application is to be monitored by this node application,
- and what other node application is to monitor this node application.

These monitoring relationships are entered in the cluster configuration file. When the node application is terminated, the relationships are canceled.

Monitoring process

The availability of a node application is monitored. Heartbeat monitoring is performed using messages which are exchanged over a special connection. If errors occur during communication, the system checks whether the KDCFILE of the monitored node is still open.

Only when the result of all these checks indicates failure is it assumed that the monitored node has failed.

You can specify the following individual aspects of monitoring (in the UTM generation):

- the interval between the monitoring messages,
- the time that the application waits for a response to the message,
- the retry factor, the number of retries before level 2 of monitoring takes effect if no response is received to a message.



openUTM manual “Generating Applications”, CLUSTER statement

You configure mutual monitoring between the node applications using the following operands:

CHECK-ALIVE-TIMER-SEC=
COMMUNICATION-REPLY-TIMER-SEC=
COMMUNICATION-RETRY-NUMBER=

7.5.2 Actions performed by the node applications if a failure is detected

It is assumed that a node application has failed if the monitored application does not respond to the messages within the configured reply time and taking account of the number of retries configured and if, on the basis of the KDCFILE of the monitored application, it is then detected that this application is no longer running but was also not terminated normally.

If failure or abnormal termination of the monitored node application is detected, openUTM proceeds as follows:

- The node application is flagged as failed in the cluster configuration file and removed from the monitoring relationships.
- If you have specified a so-called failure script during UTM generation, the monitoring node application starts this script on the computer of the monitoring node application. The following data of the failed application is passed to the failure script:
 - the application name
 - the base name of the node application
 - the host name
 - the virtual host name or blanks
 - the reference name of the node application
 - the error code of the UTM dump (Term Application Reason)



openUTM manual “Generating Applications”, CLUSTER statement

To configure the failure script, specify the operand FAILURE-CMD. This operand passes a command string containing a command to be executed and any arguments.

- The monitoring node application starts a restart monitoring timer if you have configured this:



openUTM manual “Generating Applications”, CLUSTER statement

To configure the restart monitoring timer, specify the operand RESTART-TIMER-SEC. This specifies the maximum time in seconds that a node application requires for a warm start after a failure.

- If you have specified an emergency script during UTM generation, the monitoring node application starts this script if the failed node application does not become available again after the restart monitoring timer has expired. The following data of the failed application is passed to the emergency script:
 - the application name
 - the base name of the node application
 - the host name
 - the virtual host name or blanks
 - the reference name of the node application
 - the error code of the UTM dump (Term Application Reason)



openUTM manual “Generating Applications”, CLUSTER statement

To configure the emergency script, specify the operand EMERGENCY-CMD.

This operand passes a command string containing a command to be executed and any arguments.

Sample script on detection of a failure

Sample failure and emergency scripts are supplied with openUTM. These examples output the parameters passed when they are called. If you wish to use the samples in a live environment, you must adapt them to suit the requirements of the relevant cluster.

Unix and Linux systems

The following sample scripts are supplied in the library *utmpath/shsc*:

- `utm-c.emergency`
- `utm-c.failure`

Windows systems

The following sample scripts are supplied in the directory *utmpath\shsc*:

- `utm-c.emergency.cmd`
- `utm-c.failure.cmd`

7.5.3 Application data after abnormal termination of a node application

UTM cluster applications involve application data that is valid globally throughout the cluster as well as application data that is specific to the node:

- Application data that is valid globally throughout the cluster includes GSSB, ULS and the service data of non-node-bound services. This data is present in the UTM cluster files.
- Data that is applicable locally at node level such as, for example, TLS and the service data of node-bound services (see "[Service restart](#)") is saved in the KDCFILE of the relevant node application.

The abnormal termination of a node application has the following consequences for the application data:

- Any locks that were set for the cluster's global ULS and GSSB storage areas at the time the node application terminated are retained.
- Any users who were signed on exclusively at the node application at that time continue to be signed on.
- It is not possible to access the service data of users who were signed on at the node application at the time of the failure until such a warm start is performed.
- The pages in the cluster page pool that were reserved by the abnormally terminated node application continue to be occupied.
- No node updates, cluster updates or online imports are possible.

Therefore, a warm start should rapidly be performed for abnormally terminated node applications.

7.5.4 Measures taken when a node application has been terminated abnormally

This section describes what users should do following the abnormal termination of a node application and what measures the administrator of the UTM cluster application can perform in such cases.

7.5.4.1 Measures taken for users

Users who were signed on at the node application at the time it terminated abnormally or who possess an open service bound to this node application can sign on at another node application. In this case, any open service for such a user is lost. An open service can only be continued when the user sign-on is performed **after** a warm start of the abnormally terminated node application.

However, such users cannot sign on at another node application until the abnormal termination of the node application has been detected:

- The abnormal termination of the node application has already been detected:

Users with RESTART=NO can sign on at another running node application. Users with RESTART=YES can sign on at another running node application if the application has been generated with CLUSTER ABORT-BOUND-SERVICES=YES and the user does not have a node-bound service with a transaction in the state PTC.

- The abnormal termination of the node application has not yet been detected:

The attempt to sign on is rejected until the monitoring node has detected the failure. As soon as the failure has been detected, processing continues as in the first case above.

7.5.4.2 Measures to be taken by the administrator

Depending on whether the node application can be restarted on the same node or not, the following measures may be necessary to prevent data loss:

- If the node application can be restarted on the same node after the failure, it is possible to continue working with the previous data without any problems. A failure script can, for instance, initiate an automatic restart of the node application.
- The following alternatives are available if it is not possible to restart the application on the same node, for instance if the computer has failed:
 1. Move the node application to a spare computer with the same host name / IP address. It is then possible to restart the node application on this new computer without the need to take any further measures.
 2. Move the node application to a spare computer with an identical virtual host name/IP address. Before the node application can be started on this new computer, the administration functions must be used to change the host name of the failed node in the cluster configuration file to the host name of the spare computer. After this has been done, it is possible to restart the node application on this new computer.
 3. Perform a node recovery, see section "[Node recovery](#)".

7.5.4.3 Node recovery

If it is not possible to perform a warm start for an abnormally terminated node application at the node's own node computer in reasonable time and also no virtual host has been defined then a node recovery can be performed for this node on another node in the UTM cluster in order to avoid impairing the performance of the running UTM cluster application.

Prerequisites for the use of node recovery

Node recovery requires the presence of SYSLOG files with node-specific names that can be accessed throughout the cluster.

You can dynamically generate the start parameter file with the required node name for the NODE-TO-RECOVER start parameter.

Alternatively, you can provide, for each node in the cluster, a previously set up start parameter file for node recovery that can be accessed throughout the cluster.

If you have departed from the recommendations and installed UTM or other runtime components under different paths on the individual cluster nodes and if this code is loaded from shared objects, then you should note the following:

1. For it to be possible to call node recovery, the application must have been appropriately linked for this type of use.
2. In addition, you must set the environment variables `$LD_LIBRARY_PATH` and, if necessary, `$LD_LIBRARY_PATH64` to the locally accessible paths, i.e. as they are at the start of the local node application.

For more detailed information, see the sections [“Installing the UTM runtime components for Unix and Linux systems”](#) and [“Installing further runtime components for Unix and Linux systems”](#).

Starting node recovery

Node recovery is controlled via the start parameters listed below.

NODE-TO-RECOVER

selects a node in the UTM cluster application for which node recovery is to be performed.

RESET-PTC

specifies whether or not transactions in the PTC state are to be reset on node recovery.

For a more detailed description of these start parameters, see [section “Start parameters for openUTM”](#).

Calling utmmain for node recovery

-
- > Start the program *utmmain* as a background process (see ["Starting a UTM application on Unix and Linux systems"](#) for Unix and Linux systems, ["Starting a UTM application on Windows systems"](#) for Windows systems) in order to start node recovery.

When doing this, specify the *filebase* name of the node application for which node recovery is to be performed as the first argument and start node recovery in this *filebase* directory.

! CAUTION!

The start procedure for starting the node application that is to perform node recovery must not contain any commands that have an effect on node applications running in parallel on this node computer. This includes, for example, a call of the utility program *kdcrem* prior to the start of *utmmain*.

Messages

When node recovery is started, the message K192 is sent to *stdout* and *stderr*. This message logs the values of the start parameters NODE-TO-RECOVER and RESET-PTC together with the current computer name.

A K193 message is output for every detected transaction with the PTC state, irrespective of the value of the RESET-PTC parameter.

A K160 message is output for every transaction that is reset.

At the end of node recovery, a K194 message is output which indicates the number of GSSB and ULS areas still locked by this node.

7.6 Online import of application data

After a node application has been terminated normally, messages to (OSI-)LPAPs, LTERMs, asynchronous TACs or TAC queues and open asynchronous services can be imported from the terminated node application into a different, running node application. For this to be done, their KDCFILE must originate from the same UTM generation run. Data that is imported is deleted from the terminated node application.

Online import is only possible in UTM-S applications (UTM Secure) and must be initiated using the administration functions, e.g. via WinAdmin or WebAdmin.

Imported messages are treated in the same way as newly generated messages, i.e. they are appended to the end of the queue rather than being inserted in an existing message queue on the basis of their generation times.

The following data are not imported:

- Asynchronous messages to a TAC whose queue level (QLEV) has been reached. This also applies if the TAC is generated with QMODE = WRAP-AROUND. This ensures that the import operation does not delete any asynchronous messages in the importing application.

7.7 Administering a UTM cluster application

You can administer the node applications of the UTM cluster application together:

- WinAdmin/WebAdmin

WinAdmin and **WebAdmin** provide administration functions that you can apply globally to all node applications in a UTM cluster application. In addition, for example, WinAdmin/WebAdmin also provide summary statistics covering all running node applications. WinAdmin/WebAdmin also permit you to administer individual node applications separately.

For these reasons, it is recommended that you use WinAdmin or WebAdmin to administer UTM cluster applications.



For detailed information on administering UTM cluster applications using WinAdmin or WebAdmin, refer to the respective online Help system in WinAdmin/WebAdmin and the “WinAdmin Description” or “WebAdmin Description” document.

- Using your own administration programs or administration commands

In addition to WinAdmin/WebAdmin, there is also the possibility of administering a UTM cluster application using a programmed administration facility or using administration commands. Depending on the type of change involved, the administration job applies either globally to all node applications of the UTM cluster application or only to an individual node application.



For detailed information on the programming interface and the administration commands, refer to the openUTM manual “Administering Applications”.

Note the following when administering UTM cluster applications:

- Objects that can be created dynamically must always be deleted using the administration facilities. These objects cannot be deleted by a new UTM generation alone.
- Objects that can be created dynamically cannot be deleted immediately in a UTM cluster application. Deletion can only be delayed.
- You must generate a new KDCFILE in order to release the storage space occupied by objects for which deletion was delayed in the KDCFILE.
- You can define reserve nodes with provisional properties in a UTM cluster application. You can then modify these simply, for instance using WinAdmin or WebAdmin, to produce "real" nodes.
- You can display distributed transactions that have the PTC state and then roll back the local element of this type of transaction. This action also resets the transaction in any locally connected database.

7.7.1 Actions global to the cluster and actions local to a node

You must distinguish between actions which apply globally and actions which apply locally when administering a UTM cluster application.

Actions that are global to the cluster

Actions that are global to the cluster apply to every node application. This is irrespective of whether the node application is currently active or not. All node applications subsequently perform these changes on the basis of the administration journal (see the section [“Administration journal”](#)).

Global administrative changes can be, for example:

- changing the password for a user ID
- replacing the application program or parts of the application program during live operation
- generating objects
- deleting objects from the configuration

Actions local to the node

Actions local to the node only applied to the node application in which these actions are performed.

Administrative changes local to the node can be, for example:

- terminating an individual node application
- establishment of a connection using the administration facilities



You will find information on which actions apply globally to the cluster or locally to the node in the description of the operation codes or the data structures in the openUTM manual “Administering Applications”.

7.7.2 Administration journal

The administration journal contains a log of past global administration actions, i.e. the history of the administration actions. openUTM sets up the administration journal under the filebase name of the associated UTM cluster application the first time the first node application is started (see also the section [“UTM cluster files” \(Files\)](#)).

Like all files that are global to the cluster, the administration journal is located on a storage medium which can be accessed by all node applications (see the [section “Runtime environment”](#)). The UTM system code handles concurrent accesses via NFS4 locks.

All node applications reconstruct the administrative changes that have global application on the basis of the administration journal.

- Running applications apply these actions with minimal delay. They do this at the latest before they initiate global administration actions themselves. Depending on the load on a node, this will generally be done within a few seconds.
They are notified of the need to do so by the node application that was administered directly.
A network problem can occasionally cause this notification to be lost. For this reason, and depending on the CHECK-ALIVE-TIMER-SEC operand of the CLUSTER statement, the administration journal is checked at regular intervals by the running node applications.
- Node applications that are subsequently started apply the changes during the startup phase.

7.7.3 Reducing the number of nodes

You can reduce the number of nodes in the cluster without having to modify the generation of the UTM cluster application.

To do this, proceed as follows

1. Shut down the node applications of the nodes that you want to remove from the cluster for an extended period.
2. At a node application that is still running, perform an online import for the terminated node applications, see also [section “Online import of application data”](#).

7.8 Shutting down a UTM cluster application

You have a number of different options for terminating the UTM cluster application:

- Shut down one node application, for instance using the command `KDCSHUT GRACE`.
- Shut down all running node applications of the UTM cluster application, for instance using `KDCSHUT GRACE, SCOPE=GLOBAL`.



openUTM manual “Administering Applications”, Administration command `KDCSHUT`

- Using WinAdmin/WebAdmin:
Terminate an individual node application or terminate a UTM cluster application with all running applications.



„WinAdmin Online Help“ or WebAdmin Online Help, Terminating an application

- Using an administration program you have created yourself:
Terminate an individual node application or terminate a UTM cluster application with all running applications.

If only one node application is running, shutting down this last node application has the same effect as shutting down the complete UTM cluster application.

7.9 Use of openUTM revision levels in the UTM cluster application

You can always deploy openUTM revision levels during live operation without the need to terminate the UTM cluster application. Some of the node applications can continue to run while the revision level is being installed for the remaining node applications.

To do this, the node applications must be terminated one after another and then restarted using the new revision level.

Procedure on Unix and Linux systems

On Unix and Linux systems, UTM revision levels are always installed in a directory specific to the given revision level, i.e. the files belonging to two different revision levels are stored separately and are therefore unable to overwrite each other.

For this reason you must always adjust the values of the \$UTMPATH environment variables before you start an application with a new revision level. This also applies to other environment variables that are dependent on \$UTMPATH, such as \$PATH or \$LD_LIBRARY_PATH.

Proceed as follows:

1. Install the new revision level.
2. Terminate the node application.
3. Adjust the \$UTMPATH environment variable to point to the new installation, i.e. adapt the start script as necessary.
4. Create the `utmwork` application program again if this is necessary for using the revision level.
5. Restart the node application.
6. Repeat steps 1 through 5 for all other node applications of the UTM cluster application.
7. Uninstall the old revision level if it is no longer required.

Procedure on Windows systems

On Windows systems, you can install a revision level under the same path as a previous revision level. In this event, the files of the existing version are overwritten. If this is done, installation is only possible if the UTM application has been terminated, because some of the files will otherwise be locked by the system and can therefore not be overwritten.

i Note also that this old installation cannot additionally continue to be used by other UTM applications, e.g. by additional test applications.

If you install UTM revision levels in a directory specific to the given revision level, the files belonging to two different revision levels are stored separately and are therefore unable to overwrite each other.

In this event you must adjust the values of the %UTMPATH% environment variables before you start an application with a new revision level. This also applies to other environment variables that are dependent on %UTMPATH%, such as %PATH%.

Proceed as follows:

1. Terminate the node application running the old revision level.

-
2. Install the new revision level.
 3. Reboot the computer if necessary.
This step is only necessary if the installation path has been changed, because the installation routine adjusts the %UTMPATH% environment variable in the system environment.
 4. Create the `utmwork.exe` application program again if this is necessary for using the revision level.
 5. Restart the node application.
 6. Repeat steps 1 through 5 for all other node applications of the UTM cluster application.
 7. Uninstall the old revision level if it is no longer required.

Notes on running the application as a service:

Proceed as follows if you have changed the installation path:

1. Uninstall the old version of the service before starting the system.
2. Reboot the system.
3. Install the new version of the service in the system.

7.10 Diagnostics in a UTM cluster application

Every node application writes a separate set of log files and diagnostic files. At least the log files of the node application in which a concrete error has occurred are therefore always required for debugging.

Node monitoring messages

The monitoring node application issues message K169 when monitoring starts.



For detailed information on the messages, refer to the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”.

Diagnostics documents

The following files are required for debugging cluster problems in addition to the usual documents:

- all UTM cluster files
- in the case of problems relating to the administration of the administration journal at global cluster level
- in the case of problems caused by the interaction between node applications, the log files of all node applications
- the start procedure and the procedures specified as EMERGENCY-CMD and FAILURE-CMD on UTM generation

8 Working with a UTM application

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

1. Sign on to the UTM application
A user can only sign on via clients for which LTERM partners, LTERM pools, or OSI-LPAP partners have been generated in the UTM application; see the openUTM manual "Generating Applications". It is not possible to sign on using remote login mechanisms, for example.
2. Call services of the UTM application
openUTM offers a separate authorization concept for data access control, see "[Authorization concept of openUTM](#)".
3. Enter UTM user commands if necessary.
4. Sign off from the UTM application.

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

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

8.1 Sign-on process with user IDs

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

A user can sign-on using the following client access points:

- terminals (see below)
- UPIC clients and TS applications ("[Sign-on process for UPIC clients and TS applications](#)")
- OSI TP partner ("[Sign-on process for OSI TP partners](#)")
- HTTP clients ("[Sign-on process for HTTP clients](#)")
- via the internet using WebServices (WS4UTM) ("[Sign-on process in the Internet via WebServices \(WS4UTM\)](#)")
- via the internet using WebTransactions ("[Sign-on process in the Internet via WebTransactions](#)")

Under certain circumstances, it is also possible for several users to sign on under the same user ID, see [section "Multiple sign-ons under one user ID"](#).

8.1.1 Standard sign-on process for terminals

In order to work with a UTM application via a terminal, the user starts the dialog terminal process (see [section “Starting the dialog terminal process by the user”](#)). The user also signs on to the UTM application at the same time.

Only then can the user start services and carry out interactive processing, see [section “Calling UTM services”](#).

You can use the environment variable LANG to affect the behavior of the application:

LANG is used to specify the language in which the UTM messages are output. By default, the settings LANG=De... for the German UTM messages and LANG=En... for the English UTM messages are supported. The messages are structured from NLS message catalogs.

8.1.1.1 Starting the dialog terminal process by the user

The user must start the dialog terminal process after he or she has signed on to the operating system.

- The user enters the following command on Unix and Linux systems to start the process:

```
utmpath /ex/utmdtp [-S[username]] [-Aapplicationname] [-Pptermname] [-D]
```

It is also possible to have the dialog terminal process started by Unix and Linux systems (see "[Starting the dialog terminal process through Unix and Linux system](#)").

- On Windows systems, proceed as follows:

- First set the correct *utmpath* if necessary:

```
SET UTMPATH= utmpath
```

- Start a prompt window and enter the following command:

```
utmpath \ex\utmdtp [-S[username]] [-Aapplicationname] [-Pptermname]
```

The command prompt may only be closed after the user has signed off from the application.

Legend

The specifications in square brackets represent switches that can be specified but do not have to be specified. The switches are explained below:

-S[*username*]

Using this switch, you interactively control the sign-on check (system access control) carried out by openUTM after a connection has been successfully established to the UTM application.

Dialog terminal process **with** -S switch:

If you start the dialog terminal process with the -S switch, you must transfer a UTM login *username* to openUTM for the sign-on check. With -S*username*, you can specify the UTM login directly at the start of the dialog terminal process. If -S is specified without *username*, then openUTM interactively requests the UTM login after the connection has been established.

If you specify a UTM login for which a password is generated, openUTM queries the corresponding data interactively; see the description starting on "[Standard sign-on dialog](#)".

Dialog terminal process **without** -S switch:

If you start the dialog terminal process without the -S switch, then the dialog terminal process passes the system login for the authorization check. The password is not transferred to openUTM. A password can be assigned to the login in the UTM application; the user is then requested to enter this password, as when the login is specified explicitly.

If the check on the system login is negative, then an explicit authorization dialog is run as when the -S switch is used.

A description of the authorization dialog can be found in [section "Standard sign-on dialog"](#).

i The -S switch and *username* form a string and must not be separated by any character.

-A*applicationname*

You use this switch to specify the application with which you want to be connected. *applicationname* is the name of the application generated in MAX APPLNAME. If -A*applicationname* is **not** specified at the start of the dialog terminal process, the user is asked to enter the application name.

i The `-A` switch and *applicationname* form a string and must not be separated by any character.

`-P`*ptermname*

ptermname is the name of the terminal through which the user establishes the connection to openUTM. This name must be generated in a PTERM statement or an LTERM pool must be defined for local terminals (TPOOL PTYPE=TTY, LTERM=*ltermprefix*,NUMBER=*number*). If such an LTERM pool is generated, the `-P` switch can always be omitted. Otherwise the following is true:

- Unix or Linux systems

If there is no such LTERM pool generated on Unix or Linux systems, the `-P` switch can be omitted, except for in the case described below, because on Unix and Linux systems openUTM uses the last part of the output of the `tty` command as the *ptermname*. This refers to the term after the last slash, and corresponds to the output of `basename `tty``.

The `-P` switch is only necessary when the default *ptermname* assignment by openUTM is not unique, e.g. when there are several pseudo-terminals where the last term of the `tty` command (after the last slash) are the same.

Example

The `ttys /dev/pts/12` and `/dev/inet/12` both exist at the same time. These `ttys` are generated with PTERM `pts/12` and PTERM `inet/12`. In this case the user must start the dialog terminal process with the `-P` switch as follows, for example:

```
utmpath /ex/utmdtp ... -Ppts/12.
```

- Windows systems

If there was no LTERM pool generated on Windows systems, then the user must always specify `-P` *ptermname* because openUTM uses the value `tty nnnnn`, where `nnnnn=PID`, by default on Windows systems for the *ptermname*.

Example:

If the local terminal is generated with PTERM `console,PTYPE=TTY,...`, the user must start the dialog terminal process with `utmpath\ex\utmdtp ... -Pconsole`.

i The `-P` switch and *ptermname* form a string and must not be separated by a character of any kind.

`-D` This switch only applies to Unix and Linux systems.

With this switch, the user determines the response of the dialog terminal process to the DEL key on Unix and Linux systems.

If the `-D` switch is specified: The DEL key is ignored by the dialog terminal process.

If the `-D` switch is **not** specified: Pressing the DEL key disconnects the user from the application and results in the termination of the dialog terminal process.

Example

The user wants to sign on to the `sample` application under the UTM login `user1`; the DEL key is to be ignored. The user must therefore enter the following command:

```
utmpath/ex/utmdtp -Suser1 -Asample -D
```

8.1.1.2 Starting the dialog terminal process through Unix and Linux system

A Unix or Linux system can also start the dialog terminal process after the user has successfully signed on to Unix or Linux system. To this end, the command for starting the dialog terminal process can be entered as follows, for example, in the user's `.profile`:

- `.../utmdtp switch` means that the shell remains active after `utmdtp` has terminated.
- `exec .../utmdtp switch` means that the shell is also terminated when `utmdtp` is terminated.

A second option is for the system administrator to enter the dialog terminal process as a program name in the `/etc/passwd` file for the user, or (if switches are used) start a shell procedure containing the call of the dialog terminal process.

It only makes sense to start the dialog terminal process after the work process has reported the successful cold start or warm start of the application. Otherwise, the dialog terminal process terminates again with the UTM message:

```
U111 UTM application applicationname not started.
```

If an error occurs while the dialog terminal process is executing, the process terminates with the following UTM message. The error numbers *nnnn* are explained in the openUTM manual "Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems":

```
U120 utmdtp process terminated with error number nnnn.
```

8.1.1.3 Standard sign-on dialog

The standard sign-on dialog is always performed when the following two conditions are met:

1. Automatic KDCSIGN (= automatic sign-on check) is not generated for the terminal (see "[Automatic KDCSIGN](#)").
2. No sign-on service is generated for the standard application name (generated in MAX APPLNAME) under which the user signed on (see "[Sign-on process with sign-on services](#)").

In the standard sign-on dialog, openUTM carries out a sign-on check (system access control). The sign-on dialog cannot be modified.

Different levels of freedom can be defined for the sign-on check. An overview of all options can be found in the diagram in the section "[Scenarios for the UTM sign-on check](#)".

openUTM conducts the sign-on check interactively with the user if the `-s` switch was specified at the start of the corresponding dialog terminal process. In this case, openUTM requests the login and, if generated, requires that you specify a password.

If the `-s` switch is not specified, openUTM performs the sign-on check with the system login under which the user signed on to Unix or Linux systems. In this case, a password may also be generated for the login in the UTM application, and the user will be asked to enter this password during the sign-on check.

i Please note, however, that a user cannot work simultaneously with several dialog terminal processes under one login.

Entering the password

If a **password** is generated for the user id (KDCDEF statement `USER...,PASS=password[, DARK]`), the password is always entered in a field without blanking.

With every sign-on to the UTM application, the user has the option of entering a new password to replace the previous one, provided the minimum validity period allows the password to be changed at this time. The new password must then be entered once in a field without blanking. openUTM checks the old password entered and, if necessary, the new password. If the old password is incorrect, a UTM message is output to inform the user and request again to enter a user ID.

Validity period of the password

When generating the user ID, you can define a maximum and minimum validity period for the password:

```
USER ...,PROTECT-PW=(...,maxtime,mintime).
```

The minimum validity period means that the user cannot change his or her password until this period has expired. The maximum validity period means that the user must change the password within the specified period.

If the password will become invalid within the 14 days following the sign-on procedure, openUTM warns the user in a K-message as long as a change is allowed at this time according to the minimum validity period for the password. A password can be changed as described under "[Entering the password](#)".

To prevent users that have not worked with the application for a long time from forgetting to change their password and then consulting the system administrator, the UTM application can be configured such that these users may sign on one more time after their password has expired, see section "[Grace sign-on](#)" below.

When the password is changed, openUTM checks the following:

-
- when a maximum validity period is specified, whether the new password differs from the old one.
If a password history is generated (SIGNON ...,PW-HISTORY=*n*), the new password is also checked against the last *n* passwords.
 - whether the new password corresponds to the level of complexity generated for the user ID (USER ...,PROTECT-PW=).
 - whether the length of the password is greater than or equal to the generated minimum length (USER ..., PROTECT-PW=).

If the new password fulfils all of these conditions, openUTM changes the password. The validity period of the new password again corresponds to the generated value.

If the new password does not satisfy one of these conditions, the following UTM message is output to ask the user to reenter the KDCSIGN information using the old password:

```
K097 Input for new password cannot be used - please sign on
```

If the validity period of the password has already expired when the sign-on attempt is made and if no grace sign-on is generated, the sign-on attempt is rejected with the following UTM message:

```
K120 Password expired - please sign on
```

It is then not possible to sign on again to the UTM application under this user ID until the UTM administrator has assigned a new password to the user ID.

Grace sign-on

If the validity of the password has already expired when the user attempts to sign on and if the application is generated with grace sign-on (SIGNON ...,GRACE=YES), a K message informs the user that his or her password is no longer valid. The user is also asked to enter the old password and a new password.

Scenarios for the UTM sign-on check

The following diagram shows the possible variants of the UTM-sign-on check, depending on the KDCDEF generation. If incorrect data is entered, openUTM outputs a specific UTM message and asks the user to reenter the information. If several unsuccessful sign-on attempts are made in succession from a particular terminal or under a particular user ID, openUTM outputs UTM message K094 with the standard destination SYSLOG (system log file). The maximum permitted number of unsuccessful sign-on attempts before UTM message K094 is initiated can be defined in the UTM generation with the KDCDEF statement SIGNON ... SILENT-ALARM=. A MSGTAC program unit can respond to this UTM message.

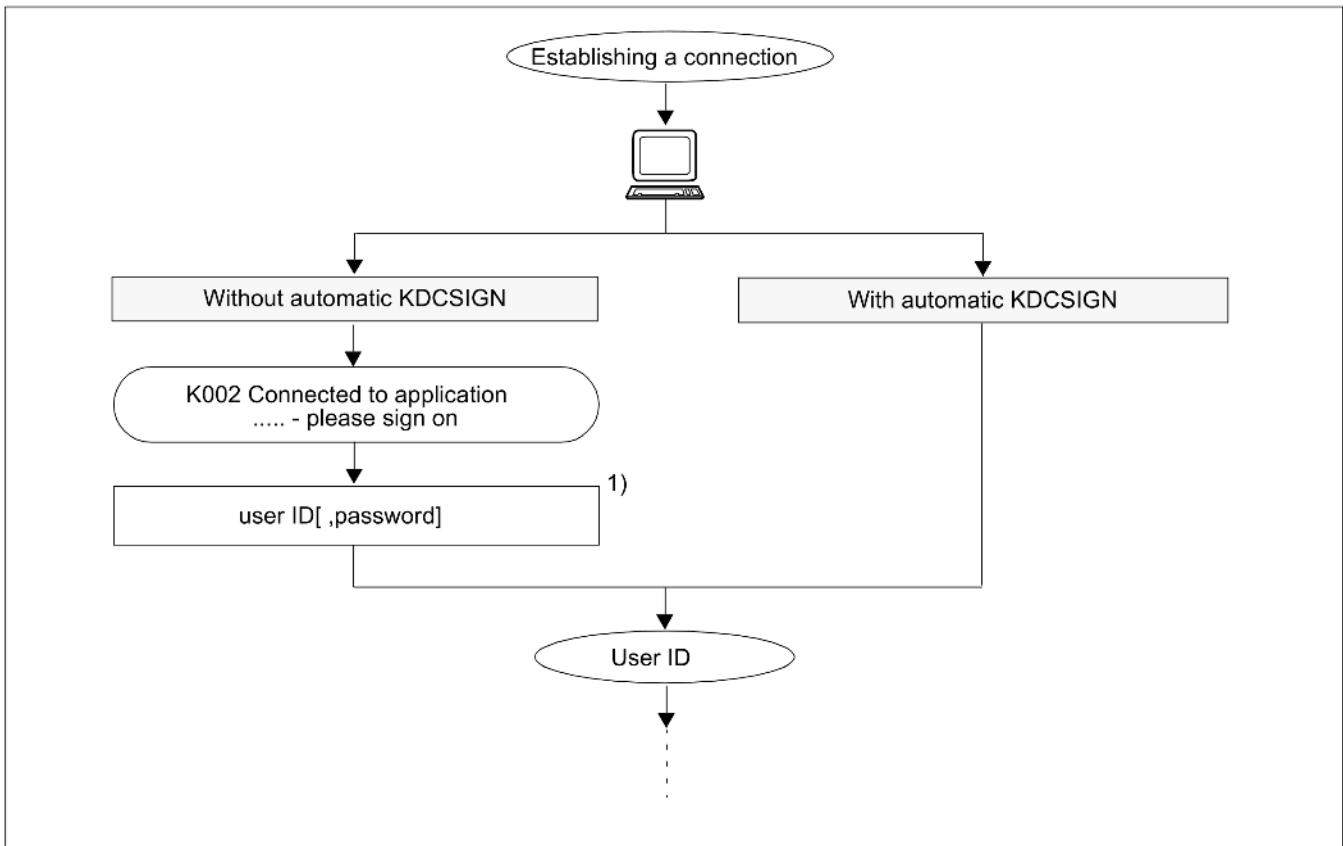


Figure 3: Sign-on check scenarios for applications with logins (part1)

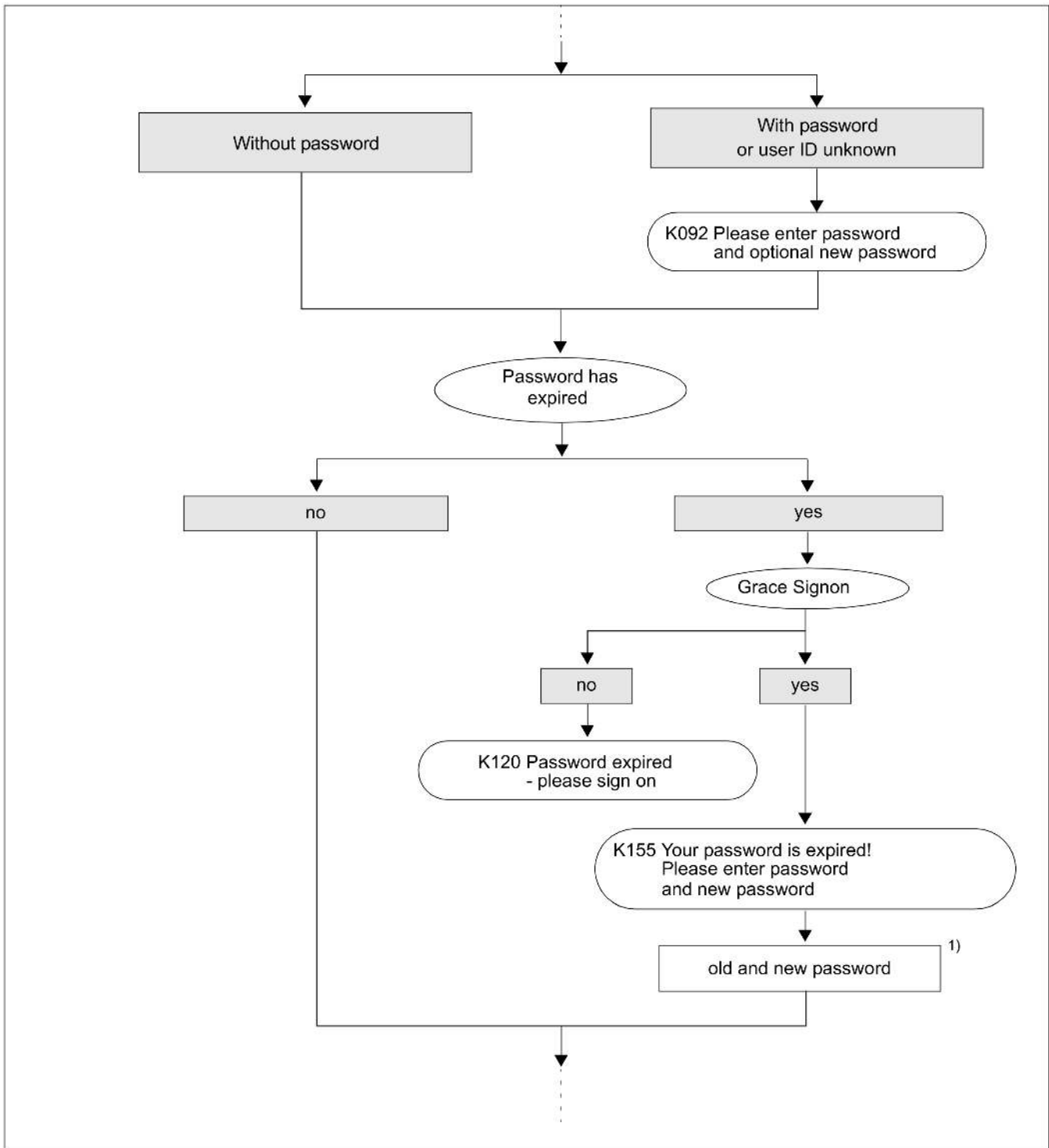


Figure 4: Sign-on check scenarios for applications with logins (part 2)

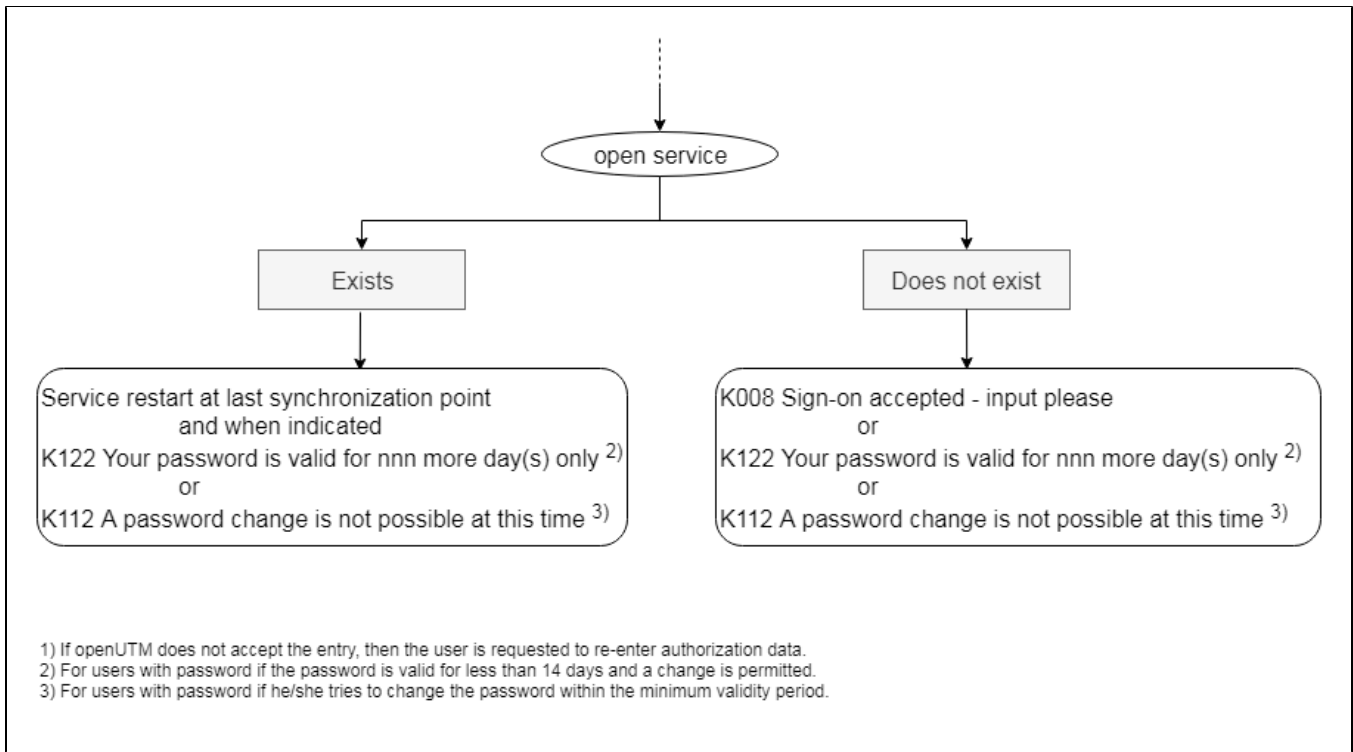


Figure 5: Sign-on check scenarios for applications with logins (part 3)

8.1.1.4 Automatic KDCSIGN

If the KDCDEF statement LTERM...,USER=*username* was specified for a terminal, after the connection setup openUTM behaves as though the user had already had already given his user ID. If a password must be entered for this user ID, openUTM requests this input from the user.

After KDCOFF BUT has been entered, it is also possible to work at this terminal under a different ID (see [section "Signing off from a UTM application"](#)).

8.1.2 Sign-on process for UPIC clients and TS applications

UPIC clients and TS applications are clients that were generated with partner type UPIC-L, UPIC-R, APPLI or SOCKET, and in case of partner type SOCKET communicate via UTM Socket Protocol (USP).

The connection is set up by the client in the case of UPIC clients, and by the client or UTM in the case of TS applications; connection setup by UTM is only possible if the TS application is generated explicitly with a PTERM statement.

If the client sets up the connection, the client must know the name or, in the case of partner type SOCKET, the port number of the UTM application as well as the host name and/or host address.

If the client establishes the connection, it must know the name or, in the case of partner type SOCKET, the port number of the UTM application and the host name and/or host address.

When a connection is set up successfully, a UPIC client or TS application signs on in two stages:

1. Implicit sign-on using a **connection user ID**

A connection user ID is strictly assigned to an LTERM partner of a TS application or a UPIC client and is created explicitly or implicitly at UTM generation:

- Explicit creation by USER= specification in the LTERM statement.
Additional characteristics can be defined with the KDCDEF statement USER for connection user IDs defined in this way.
- Implicit creation by openUTM if no USER was specified in the LTERM statement or if an LTERM pool is used (TPOOL statement). The LTERM name is then used as the name of the connection user ID; in the case of an LTERM pool, the LTERM name is made up of the generated prefix and a serial number, e.g. UPIC0025. For LTERM pools, special key codes can be assigned to the connection user ID with TPOOL ... USER-KSET=. The access options of the connection user ID can thus be restricted.

If no sign-on attempt is made under a real user ID, the preliminary sign-on of the connection user ID becomes permanent. This is recorded with a message. In the case of UPIC clients, this message is also output if the client subsequently signs on under a real user ID.

2. Explicit sign-on using a **real user ID** (optional)

UPIC clients and TS applications behave differently in this case:

- In the case of UPIC clients, the user ID and sign-on data must be set in the respective UPIC interface calls. UPIC then transfers these values to openUTM, which then performs the sign-on for the transferred user ID. This replaces the connection user ID for the duration of the conversation.

At the end of the conversation, the user is signed off again if the user ID is generated with SIGNON OMIT-UPIC-SIGNOFF=NO.

If the user ID is generated with SIGNON OMIT-UPIC-SIGNOFF=YES, the user remains signed on after the end of the conversation and is only signed off,

- if another user ID is passed in the UPIC protocol before a new UPIC conversation is started over the same UPIC connection,
- or when the connection is cleared.

If the UPIC client does not transfer any sign-on data in the UPIC interface calls, signing on using a real user ID is only possible with a corresponding sign-on service; see ["Sign-on process with sign-on services"](#).

-
- A user can only sign-on under a real user ID via a transport system connection if an appropriate sign-on service is generated for the application; see "[Sign-on process with sign-on services](#)". It is not possible to sign on with a real user ID using the standard sign-on dialog.

If a TS application signed on using a real user ID, this user ID replaces the connection user ID for the full duration of the connection.

In the case of both UPIC clients and TS applications, the connection user ID remains signed on for at least as long as the real user ID. If the connection is lost, a renewed connection setup attempt may be rejected if a program is still running under the real user ID and the connection user ID is thus also considered to be signed on. In this case, the user must wait until the program terminates before signing on again.

8.1.3 Sign-on process for OSI TP partners

In order for an OSI TP partner to sign on to the UTM application, the partner must know the the address of the OSI TP access point of the UTM application. This data is configured in the OSI TP partner.

In the case of OSI TP partner, the connection setup initiative can come from either the partner or openUTM. This means that several parallel connections, known as associations, can be established via one logical connection. An association name is assigned to each association.

Following a successful connection setup, the client first signs on under its association name. This name is made up of the name specified in OSI-LPAP ...,ASSOCIATION-NAMES= and a serial number, e.g. ASSOC03.

Once an application context containing the abstract syntax UTMSEC has been generated for OSI TP communication between the two partners (in case of openUTM in the APPLICATION-CONTEXT parameter of the OSI-LPAP statement), the communication partner can pass a genuine user ID and authorization data in the relevant log fields. openUTM then signs on with the user ID that has been passed. This sign-on then applies for the duration of the OSI TP dialog. The user is then signed off again at the end of the OSI TP dialog.

If no genuine user ID is passed, the client remains signed on under its association name.

8.1.4 Sign-on process for HTTP clients

HTTP clients are clients that were generated with PTYPE=SOCKET and communicate using the HTTP protocol.

For an HTTP client to be able to log on to the UTM application, it must know the address of the transport system endpoint of the UTM application. With HTTP clients, the initiative to establish a connection always comes from the client, and an HTTP client sends an HTTP request to the transport system endpoint.

Once the connection has been successfully established, an HTTP client is logged on in two steps:

1. Implicit logon using a **connection user ID**

A connection user ID is permanently assigned to the LTERM partner of an HTTP client and is generated explicitly or implicitly during UTM generation:

- Explicitly by specifying USER= in the LTERM statement.
Additional properties can be defined for a connection user ID defined in this way using the KDCDEF statement USER.
- Implicitly by openUTM, if no USER was specified in the LTERM statement or if it is a LTERM pool (TPOOL statement). The LTERM name is then used as the name of the connection user ID; for a LTERM pool, the LTERM name is made up of the generated prefix and a sequential number, for example, HTTP0025. For LTERM pools, special key codes can be assigned to the connection user ID with TPOOL ...USER-KSET=. This allows you to restrict the access options of the connection user ID.

If no login under a real user ID follows, the temporary login of the connection user ID becomes a final one. This is logged with a message.

2. Explicit logon using a **real user ID** (optional)

In an HTTP request, an HTTP client can send authentication data to the UTM application in the Authorization header field. If this header is specified, the value must begin with "basic " and <userid>:<password> must be base64 encoded. Otherwise the request will be rejected with status code "400 invalid http header format". openUTM checks the passed authentication data. If the check fails, openUTM rejects the login with "401 user not authorized". Otherwise the user is logged on and can call the services of the application, see "[Starting services from the HTTP client](#)".

The UTM application may also require the HTTP client to send authentication data. This can be configured either for all HTTP requests received via a transport system endpoint by setting the USER-AUTH parameter in the KDCDEF statement BCAMAPPL to *BASIC, or for the path specification of an HTTP request by setting the USER-AUTH parameter in the KDCDEF statement.

8.1.5 Sign-on process in the Internet via WebServices (WS4UTM)

A service of the UTM application can be called from WebService clients using WS4UTM. This allows the user to access certain services of a UTM application over the Internet.

The WebService client can be used to structure the sign-on process via WS4UTM:

1. The user specifies a WebService name and a method in his/her WebService client. The WebService is permanently linked to a UTM application by the configuration. A connection to the UTM application is established over UPIC.
The WebService client may possibly execute an intermediate dialog, for instance to obtain proof of authorization.
2. As when using a terminal, the user may have to specify their UTM user ID and password. Whether or not the user has to go through an authorization dialog of this type and the appearance of any such dialog will depend on the way in which the WebService client is structured. It is, for instance possible to "hide" the UTM user ID /password in the WebService client or to specify it within the configuration of the WebService, with the result that the authorization dialog is handled internally.
3. The job data (TAC and user data) is sent together with the authorization data to a WebService server via http /Soap and then to the UTM application over the UPIC connection. The UPIC connection is cleared again after the response has been returned to the WebService client.

The Apache Axis server is used as the WebService server.

Communication takes place over Apache Tomcat and Axis using Soap messages and the http protocol. WS4UTM uses the UPIC interface in openUTM in order to connect to the UTM application.



For further information, refer to the manual "Web-Services for openUTM".

8.1.6 Sign-on process in the Internet via WebTransactions

A UTM application can be connected to the Internet via WebTransactions. This means that a user can access the services of a UTM application using a browser.

Signing on via WebTransactions can be configured using the WebTransactions application:

1. The user enters the URL of the WebTransactions application in the browser. The connection is then established to the UTM application. The WebTransactions application might output an intermediate dialog box, e.g. to verify authorization for accessing the WebTransactions application.
2. The user may have to specify the UTM user ID and password (if necessary), as with a terminal. However, the actual configuration of the WebTransactions application determines the format of the sign-on dialog and whether or not the user must complete such a dialog. For example, it is possible to “hide” the UTM user ID /password in the WebTransactions application so that the sign-on dialog runs internally and the user is signed on immediately the URL is entered.

The user can then call the services of the application, see ["Calling UTM services"](#).

To connect to the UTM application, WebTransactions uses the UPIC interface of UTM. More details can be found in the WebTransactions manual “Web Access to openUTM Applications via UPIC”.

8.1.7 Multiple sign-ons under one user ID

If the user ID was generated at KDCDEF generation with RESTART=NO and the UTM application with the default value MULTI-SIGNON=YES, a user can be signed on to the UTM application via different connections – though only once via a connection to a terminal. Multiple sign-ons are only possible for real user IDs, **not** for connection user IDs. More details on connection user IDs can be found on ["Sign-on process for UPIC clients and TS applications"](#).

If a user signs on under a user ID generated with RESTART=YES via an HTTP client or an OSI TP partner with the functional unit “commit” selected for its conversation, a further sign-on is possible under this user ID because openUTM does not restart the service in this case and the user ID thus behaves as though no restart was generated.

The same applies if the user signs on via an OSI TP partner and executes an asynchronous request.

Otherwise, a user can only be signed on once at any one time under a user ID generated with RESTART=YES, because the resources needed to restart the service are assigned to the user ID.

Preventing multiple sign-ons for user IDs with RESTART=NO

The MULTI-SIGNON parameter of the SIGNON statement can be used at UTM generation to define that a user can only be signed on to openUTM once at any one time regardless of the restart attribute.

However, this definition does not apply to sign-ons via an OSI TP partner for the execution of asynchronous requests.

8.1.8 Sign-on process with sign-on services

Sign-on services, also known as SIGNON event services, are user-programmed services that can be used to define your own sign-on processes. Sign-on services can be used by terminals, UPIC clients, and TS applications, i.e. by clients generated with the PTERM or TPOOL statement (except HTTP clients).

Calling sign-on services

A sign-on service is linked to the application name. If a client signs on under a particular application name, the sign-on service associated with this application name is started and replaces the standard sign-on process described in the preceding sections. If several application names are generated with BCAMAPPL statements, several different sign-on services can exist in an application. This means that client-specific sign-on services can be created, e.g. one for terminals, one for UPIC clients, and one for TS applications. More details can be found in [section “Sign-on service for terminals”](#) through [section “Sign-on service for UPIC clients”](#).

If no sign-on service is generated for an application name, the client runs through the standard sign-on process.

Generating sign-on services

Sign-on services are generated as follows; see also the openUTM manual “Generating Applications”:

- TAC KDCSGNTC is used to generate the sign-on service for the standard application name (defined in MAX APPLINAME).
- BCAMAPPL *appliname2...*,SIGNON=*signon-tac* is used to generate the sign-on service for the application name *appliname2*. *signon-tac* must be defined in a TAC statement.
- If you also want UPIC clients to be able to use sign-on services, SIGNON ...,UPIC=YES must be generated as well.

A PROGRAM statement is also needed for each of these TACs. The name of the first program unit run through in the sign-on service is specified here.

Programming sign-on services

The special KDCS calls SIGN ST, SIGN ON and PEND PS are used for programming a sign-on service. A detailed description of how to program a sign-on service and what rules to observe in the process can be found in the corresponding section of the openUTM manual „Programming Applications with KDCS”.

8.1.8.1 Sign-on service for terminals

A sign-on service for terminals is generally made up of two parts:

Sign-on service	
Part 1	Part 2
The service asks the user for identification, reads the authorization data with MGET, and transfers this data to openUTM for checking. The service is not yet assigned to any user ID.	openUTM has accepted the authorization data and has assigned the sign-on service to the established user ID.

openUTM may insert an intermediate dialog between the first and second parts of the signon process if the period of validity of the password has already expired and the application has been generated with grace sign-on. A K message informs the user that the password is no longer valid. At the same time, the user is prompted to enter the previous password and a new password.

Special cases of the sign-on service for terminals

The sign-on service must be changed accordingly for the UTM generation of LTERM partners with automatic KDCSIGN.

LTERM partners with automatic KDCSIGN

The sign-on service receives the information that the user ID is already known to the system when calling SIGN ST. An intermediate dialog can now be run to change a password whose period of validity has expired.

8.1.8.2 Sign-on service for TS applications

The sign-on service is started under the connection user ID.

The authorization data of a real user ID can be passed in the sign-on service via the SIGN ON call. If openUTM accepts the data, then the user is signed on under the specified user ID when the sign-on service ends properly. The sign-on attempt is rejected if the authorization data of the TS application is incorrect or if there is an open service under the connection user ID.

If the sign-on is unsuccessful under a real user ID, a successful sign-on under a real user ID must follow within the same sign-on service, as otherwise the connection is cleared down when the sign-on service is terminated. This means that the connection user ID is not a fallback step for a failed sign-on attempt.

If there is no user ID passed in the sign-on service, then the user is signed on permanently under the connection user ID when the sign-on service terminates properly.

8.1.8.3 Sign-on service for UPIC clients

A distinction is drawn between two possible scenarios when signing on using a sign-on service:

- The UPIC client transfers authorization data to openUTM in the UPIC protocol. If openUTM accepts the data, the sign-on service is started under the transferred real user ID and the client is signed on under this user ID, provided the sign-on service is completed successfully.
- If the UPIC client does not transfer any authorization data in the UPIC protocol, the signon service is started under the connection user ID. The authorization data of a real user ID can be passed in the sign-on service. If openUTM accepts the data, then the user is signed on under the specified user ID when the sign-on service ends properly. If no authorization data is passed, then the conversation runs under the connection user ID.

If the sign-on fails under a real user ID, a successful sign-on must follow under a real user ID, as otherwise the conversation is terminated when the sign-on service is terminated. This means that the connection user ID is not a fallback step for a failed sign-on attempt.

To ensure that client programs can be implemented regardless of whether or not the UTM application uses a sign-on service, messages from the client that were not read in the signon service can be read in the subsequent program unit after terminating a program unit of the sign-on service with PEND PA, PEND PR, PEND PS or PEND FC without preceding MPUT.

8.1.8.4 Possible applications for the sign-on service

The sign-on service offers the user a range of practical options, which are outlined below:

- TS applications can sign on to a UTM application using a sign-on service with a real user ID. They are thus integrated in the system access and data access concept of openUTM.
- A real name entered by the user can be converted into a user ID which is defined in the UTM generation (*USER username*).
- In the case of a global DB/DC authorization concept, a database call can be used in the second part of the sign-on service to retrieve the current authorization profile for this USER from the database and possibly store it in a user-specific long-term storage area (ULS).
- In the second part, the sign-on service can ask the user to change his or her password, for example because the system is monitoring the time span in which the user may use the same password.
- Statistics can be produced on all attempted and successful sign-ons.
- The sign-on service can also provide the user with useful information in the case of a subsequent service restart. Such information includes bulletins, maps of keyboard layout, or a display of the service restart. This requires an additional dialog step.
- If openUTM starts the sign-on service following a SIGN OB call (= KDCOFF BUT by program), it may be advisable to read the last input from the terminal with MGET if new authorization data was already entered there.

8.1.8.5 Properties of sign-on services

Outputting the last dialog message by the sign-on service

If there is not a service restart pending and the sign-on service is terminated with MPUT PM and PEND FI, the last dialog message of the last user's session is output if the user is generated with RESTART=YES. The user can then continue working with the same screen that was being used when the last session was terminated, regardless of whether this occurred inside or outside of a service.

Messages

If a UTM application uses a sign-on service, then the following messages are not produced (and therefore not output to the SYSLOG and MSGTAC):

K001, K002, K004 through K008.

Message K033 (Successful sign-on) is also created when a sign-on service is used.

Unsuccessful attempts in the sign-on service

In the sign-on service, unsuccessful attempts of the user to sign on can be intercepted: if openUTM does **not** accept the authorization data entered by the user, the sign-on service can ask the user to repeat the input. The maximum number of input attempts can be programmed. If this number is exceeded, the sign-on service should terminate. UTM then shuts down the connection in the case of TS applications and terminals, whereas only the conversation is ended in the case of UPIC.

In addition, openUTM counts all of the unsuccessful attempts of a client or unsuccessful attempts from a user ID made in uninterrupted succession, also over a series of sign-on services. The maximum permitted number of failed sign-on attempts must be defined in the UTM generation. After this number of failed sign-on attempts has been made (see openUTM manual "Generating Applications", KDCDEF statement SIGNON, operand SILENT-ALARM), openUTM reports this event to SYSLOG (silent alarm, UTM message K094). Sign-on attempts by unauthorized persons can be uncovered and averted with an MSGTAC routine.

Abnormalities in the sign-on service

openUTM checks whether the rules for the sign-on service are observed. This also provides protection against any manipulation of the program units of the sign-on service. If such errors occur, openUTM terminates the sign-on service with PEND ER and shuts down the connection to the terminal. The connection is then shut down in the case of TS applications and terminals, whereas only the conversation is ended in the case of UPIC.

8.1.9 Behavior in the event of locked clients/LTERM partners

Behavior for locked clients

Clients can be locked by UTM generation (PTERM...,STATUS=OFF) or administration command. Locking a client has the following effects:

- Any connection setup request will be rejected.
- Any existing connection will be retained; the lock only comes into effect if a new connection setup request is received from this client.

Behavior for locked LTERM partners

LTERM partners can be locked by UTM generation (LTERM...,STATUS=OFF) or administration command.

In the case of UPIC clients and TS applications, locking the LTERM partner has the same effect as locking the client.

In the case of terminals, locking an LTERM partner has the following effects:

- Any connection setup request will be carried out, but the following UTM message will be output after the connection has been established:

```
K027 Terminal &LTERM is locked - contact administrator or sign off.
```

- Any existing connection will be retained; the next input from the terminal will be acknowledged with UTM message K027.

8.2 Sign-on process without user IDs

openUTM does not perform a sign-on check for UTM applications for which no user IDs are generated. The clients are signed on under their LTERM names or association names. UPIC clients, HTTP clients and OpenCPIC clients are not permitted to transfer real user IDs in this case.

If the UTM application uses sign-on services ("[Sign-on process with sign-on services](#)"), an application-specific sign-on check can then be performed, e.g. using a database with authorization data.

If sign-on services are not used, the user can work with this application as soon as a connection has been successfully established to the UTM application. In the case of terminals and TS applications (clients with partner type APPLI or SOCKET that communicate via the UTM Socket Protocol (USP) in case of partner type SOCKET), the user receives a message from openUTM depending on whether an open service is still known for this LTERM partner.

- If no open service is known for the LTERM partner in the application, openUTM outputs the UTM message
`K001 Connected to application example - input please`
The user can then start services and enter UTM user commands.
- If an open service is known for this LTERM partner in the application, the output from the last synchronization point of the interrupted service is displayed on the screen and the user can continue the service. See also "Service restart" and "Screen restart" in the openUTM manual „Programming Applications with KDCS“. One of the prerequisites here is that RESTART=YES was generated for this LTERM partner. However, this also means that the user may also be able to continue the service of another user.

Note that openUTM links a service to the LTERM partner in an application without user IDs. An interrupted service can therefore only be continued from the same client, unless the assignment of LTERM partner and physical client (defined in the PTERM statement) is changed accordingly with the administration command KDCSWTCH.

If clients are locked, the behavior is the same as for user IDs; see "[Behavior in the event of locked clients/LTERM partners](#)".

! CAUTION!

In a UTM application without user IDs, all users have administration authorization.

8.3 Calling UTM services

If the UTM sign-on check runs successfully, the user is authorized to work with the UTM application, i.e. he or she can start new services (see below) or continue open services.

Sections [section “Starting services from the terminal”](#) and [section “Starting services from TS applications”](#) illustrate how new services are started for the individual client types. For a description of what happens when an open service is still known for this user ID in the application, see [section “Service restarts”](#).

8.3.1 Starting services from the terminal

Following a successful sign-on, the user can start a service by entering a transaction code (TAC) or pressing an appropriately generated function key.

Starting a service by entering a transaction code

If no sign-on service is performed, openUTM outputs the following message in line mode:

```
K008 Sign-on accepted - input please
```

The user can start a service by entering a TAC and possibly a message. The first eight characters input are interpreted by openUTM as the TAC. If the TAC is shorter than 8 characters, it must be separated from the message by a blank.

If a sign-on service is performed, the sign-on service determines the next step. The user then receives output, or a service is started immediately.

Key assignment on terminals on Unix and Linux systems

The following key assignment applies:

Key	Effect
Correction	As in the shell
return	The data is sent to the application.
END DEL	The dialog is terminated and the user is signed off from the application. The DEL key does not work if the user signed on with the -D switch.
other system keys	Must never be used as both the escape sequences of the key and the data are sent to the application.

Key assignment on Windows systems

The following key assignment applies when working with the application on Windows systems at the console using the command prompt:

Key	Effect
Correction	As in the command prompt.
return	The data is sent to the application.
END, CTRL+C	The dialog is terminated and the user is signed off from the application.
DEL	The key is ignored.
Cursor keys	As in the command prompt, i.e. history of the last entries.
other system keys	Must never be used as both the escape sequences of the key and the data are sent to the application.

Entering invalid transaction codes

If the user enters an incorrect TAC, the following message is output:

```
K009 Transaction code <tac> is invalid - input please
```

If a BADTACS dialog service is generated in the application, then the BADTACS service is started instead. After the BADTACS dialog service has ended, the user remains signed on and can start a service as described above.

8.3.2 Starting services from the UPIC client and OSI TP partner

After the connection has been set up, the OSI TP partner or UPIC clients can start conversations. To this end, the TAC is set by the client, e.g. using the *Set_TP_Name* function on the CPI-C interface or a corresponding entry in the side information file. This TAC is transferred to openUTM, possibly in conjunction with authorization data. When the sign-on check has been performed successfully, the following apply:

- In the case of OSI TP partner and UPIC clients with no sign-on service, the service associated with the TAC is started immediately.
- In the case of UPIC clients with a sign-on service, the service associated with the TAC is not started until the sign-on service has been concluded.

The user is signed off again at the end of the conversation if he or she signed on for this conversation under a real user ID. This is not valid for UPIC clients in UTM application generated with SIGNON OMIT-UPIC-SIGNOFF=YES, see "[Sign-on process for UPIC clients and TS applications](#)".

8.3.3 Starting services from the HTTP client

After the connection has been set up, the HTTP clients can start conversations. For details specifying the authorization data and the TAC please refer to the openUTM manual "Concepts and Functions" and to the openUTM manual "Programming Applications with KDCS".

8.3.4 Starting services from TS applications

TS applications (clients with partner type APPLI or SOCKET that communicate via the UTM Socket Protocol (USP) in case of partner type SOCKET) behave similarly to terminals:

- If there is no sign-on service, the TS application receives message K001 if the message destination PARTNER was assigned to this message; see the description of the KDCMMOD tool in the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”.

The TS application can then start a service by transferring a TAC, and possibly a message, to the UTM application. In this case, the first 8 characters of the message are interpreted as the TAC. If the TAC is shorter than 8 characters, it must be separated from the message by blanks.

- If a sign-on service is performed, this service determines the next step. The sign-on service can either start a service directly to send a message to the TS application. In the latter case, the next message must contain a TAC in the first 8 characters, i.e. the same applies as when no sign-on service is used (see above).

Once the service has terminated, the next service can be started.

8.3.5 Service restart

A service restart is generally performed if:

- a client signs on under a user ID that was generated with RESTART=YES,
- and an open service is still known for this user ID in the application.

If a message was sent to the client at the last synchronization point then openUTM sends this message to the client again and the user can then continue the service. Otherwise the open service is continued immediately.

Depending on the type of client and on the sign-on process involved, the following apply to the service restart:

- Standard sign-on process for terminals and TS applications:
openUTM performs the service restart automatically.
- Standard sign-on process for UPIC clients and OSI TP partner:
The client must start a specific conversation, which requests the restart using the UTM user command KDCDISP (see the manual „openUTM-Client for the UPIC Carrier System“, for example). The service cannot be restarted from OSI TP partner if the “commit” functional unit was selected.
- If an open service exists for the user and no service restart is requested, the open service is terminated abnormally.
- Signing on using a sign-on service:
The sign-on service must initiate the restart or terminate the open service abnormally.

i In an application with user IDs, a service is linked to the user ID. This means that the user can continue an interrupted service even on a different client, provided the LTERM partner of the client has the correct authorization and the client type remains the same.

8.4 Authorization concept of openUTM

In addition to system access control based on user IDs, openUTM offers a sophisticated system access and data access concept. This makes it possible to control which users can access which services of the UTM application via which LTERM partners.

You can choose between a user-oriented variant (**lock/key code** concept) and a role-oriented variant (**access list** concept). These variants are generated using lock codes, access lists, keysets, and key codes:

- A service is protected either with lock codes (lock/key code concept) or with an access list (access list concept) (TAC statement LOCK= or ACCESS-LIST=).
- A user ID receives a keyset with one or more key codes (USER statement KSET=). The key codes define the authorizations.
- An LTERM partner receives a keyset with one or more key codes, as well as lock codes if the lock/key code concept is used (LTERM or TPOOL statement, KSET= and LOCK= operands).
- Keysets are defined separately in KSET statements.

The preconditions under which users can sign on and when they can start or continue a service (following a service restart) are outlined in the following table for both concept variants.

Action	Preconditions	
	Lock/key code concept	Access list concept
Sign on via specific LTERM partner	A key code of the user ID matches the lockcode of the LTERM partner.	Sign-on is always possible.
Start a service	The user ID and LTERM partner have a key code that matches the lockcode of the TAC.	The user ID and LTERM partner each have a key code which is contained in the access list of the TAC. The key codes of the user ID and LTERM need not be identical.
Continue service (following service restart)	A key code of the LTERM partner via which the user continues the service must match the lockcode of the follow-up TAC.	A key code of the LTERM partner via which the user continues the service must be contained in the access list of the follow-up TAC.

Messages in the event of incorrect authorization

If authorization is invalid, the following messages may be output to the terminal user (a corresponding return code is supplied with the sign-on service):

```
K005 User identification user is locked - please sign on
```

If the key code of the user does not match the key code of the LTERM partner (sign-on service: return code U02).

```
K009 Transaction code <tac> is invalid - input please
```

If the user or LTERM is not authorized to start the service. If a BADTAC service is generated, the BADTAC service is started instead.

K123 LTERM does not have the rights to continue the service - please sign on

If the LTERM partner via which the user signed on at the service restart is not authorized to start the follow-up TAC (sign-on service: return code U16). This message may be output in particular if a user continues the service from a different terminal and hence a different LTERM.



More information can be found in the openUTM manual “Concepts und Functions” and the openUTM manual “Generating Applications”.

8.5 Signing off from a UTM application

The following sections describe the various ways in which a client can sign off from the UTM application or is signed off by UTM. In this case, terminals differ from all other clients because users can only sign off from the application explicitly from terminals.

Signing off in the event of a timeout

Maximum wait times can be defined at UTM generation using:

- the TERMWAIT= (PEND KP timer) and PGWTTIME= (PGWT timer) operands in the KDCDEF control statement MAX
- the IDLETIME= (transaction end timer) operand of the PTERM or TPOOL statement or the OSI-LPAP statement for OSI TP partners.

If a wait time set with these timers expires, the following message is output to terminals:

```
K021 No input within the specified period
```

openUTM then signs off the user ID and shuts down the connection to the client. The client can subsequently sign on to the application again and continue a possibly open service, see [section "Service restarts"](#).

Signing off with the KDCOFF command

A terminal user can sign off from the UTM application by entering the UTM command KDCOFF or KDCOFF BUT. See also the UTM user command KDCOFF on ["KDCOFF - sign off from a UTM application"](#).

KDCOFF from a program

openUTM offers the function calls SIGN OF and SIGN OB, which can be used to trigger the effect of the KDCOFF or KDCOFF BUT user command in a dialog program unit. SIGN OF/OB is possible for terminals, UPIC clients, and TS applications. These calls are not permitted in program units running for an OSI TP partner.

SIGN OF and SIGN OB work as follows:

SIGN call	Command	Effect
SIGN OF	KDCOFF	openUTM shuts down the connection to the client
SIGN OB	KDCOFF BUT	The connection remains open for terminals; the user is signed off. In the case of UPIC clients and TS applications, the connection is shut down (as with SIGN OF).

The call has different effects for terminals and UPIC clients/ TS applications:

- In the case of terminals, openUTM first outputs the MPUT message and message K095 to the terminals. Only with the next (arbitrary) input from the terminal is the user signed off and the connection shut down (with SIGN OF).
- In the case of UPIC clients and TS applications, the MPUT message is sent and the connection shutdown is then initiated immediately.

Some of the possible applications of the SIGN OF/OB function call are outlined below:

- Applications with particular security requirements. After signing off, a user can only process a single service.
- The control part of the screen also offers “Sign Off” or “Sign On” as possible follow-up actions. Depending on the input, the follow-up program unit then creates a SIGN OF or SIGN OB call. Following the dialog output of this program unit and the subsequent input, either the connection to the terminal is shut down or the sign-on service is started.
- Improved data protection: The screen can be overwritten by the last MPUT, no process-specific data remains on the screen.

8.6 UTM user commands for terminals

This section describes all of the UTM user commands available to the terminal user:

- KDCOUT, for requesting asynchronous messages
- KDCDISP, for requesting the last dialog message again
- KDCLAST, for repeating the last output
- KDCOFF, for signing off

8.6.1 KDCOUT - output asynchronous messages

With the KDCOUT command, the user can request the output of asynchronous messages.

openUTM announces asynchronous messages with the following message:

```
K012 nnn asynchronous message(s) present
```

The UTM message appears in the system line together with a dialog output at this terminal. The number of asynchronous messages is specified with *nnn*. The user can retrieve these messages using the KDCOUT command. If there are no messages for the terminal when KDCOUT is entered, openUTM outputs the UTM message:

```
K020 No message(s) present
```

When an asynchronous message is retrieved with KDCOUT, it is deleted by the next input, except when KDCLAST is entered (see "[KDCLAST - repeat the last output](#)").

The result of the KDCDEF statement LTERM ..., RESTART= NO is that any pending asynchronous messages are deleted when the connection is set up or shut down to this LTERM partner.

The function variants of openUTM have the following effects on the handling of asynchronous messages:

- With UTM-S applications, asynchronous messages are logged even if the application run is interrupted and are retained until retrieved with KDCOUT.
- With UTM-F applications, asynchronous messages are only stored during the application run. They are lost when the application run terminates.

8.6.2 KDCDISP - output the last dialog message

While a UTM application is running, a user can output the last dialog message once again with the KDCDISP command.

If the user enters the KDCDISP command after the sign-on service has concluded or after returning from an inserted service, openUTM redisplay the last screen of the last session or the last screen of the interrupted service.

The KDCDISP command is useful in the following situations:

- As a result of operating errors at the terminal, the screen content after a dialog output is partially or fully destroyed.
- The user has received asynchronous messages on the screen while processing a service (either requested with KDCOUT or sent automatically by openUTM) and then wants to continue the open service. In this case, the KDCDISP command is issued to redisplay the last dialog output.
- When the UTM application has been terminated and restarted, the user can (for orientation purposes) issue the KDCDISP command to repeat the last dialog output of the service concluded before the application terminated. However, this only applies with a UTM-S application and if the service restart facility was not explicitly deactivated by the KDCDEF statement `USER ...,RESTART=NO` (or `LTERM ...,RESTART=NO`, if the application was generated without user IDs).

8.6.3 KDCLAST - repeat the last output

The KDCLAST command enables you to repeat the last output message at the terminal, regardless of whether this was a dialog message or an asynchronous message.

If the last message output was an asynchronous message, this output is repeated with KDCLAST. However, the asynchronous message is thereby not yet released.

If the KDCLAST command is entered after the sign-on service has concluded, openUTM redisplayes the last screen of the sign-on service. If the command is entered after returning from an inserted service, the last screen of the inserted service is redisplayed.

8.6.4 KDCOFF - sign off from a UTM application

You can enter the UTM command KDCOFF to sign off from the UTM application. This shuts down the connection between the dialog terminal process and the UTM application and terminates the dialog terminal process.

If the dialog terminal process was started automatically by the Unix or Linux system following a successful sign-on (see "[Starting the dialog terminal process through Unix and Linux system](#)"), the dialog with the Unix or Linux system is also terminated.

If you sign off at the end of a transaction while a service is being processed, the processing is interrupted. It can be continued when you later sign on to the UTM application again, if the user is generated with RESTART=YES..

KDCOFF BUT

By entering KDCOFF BUT, you can sign off in such a way that the connection between the dialog terminal process and the UTM application is retained. It is needed for a subsequent sign-on, or the sign-on service is started.

Messages

If KDCOFF [BUT] is entered, openUTM responds by outputting one of the following UTM messages:

```
K019 Sign-off for application example accepted
```

The user entered KDCOFF or, in an application without user IDs, entered KDCOFF BUT. The terminal is no longer connected to the UTM application.

```
K018 Sign-off for application example accepted - please sign on
```

The user entered KDCOFF BUT in an application with user IDs and without a sign-on service. openUTM asks the user to sign on again with a user id. This also applies if the user signed on without the -S switch.

```
K003 Command KDCOFF is not permitted at this time.
```

The command is entered after a PEND KP call or blocking call (e.g. PGWT) of the program unit.

9 Exchanging programs during operation

openUTM offers two different methods for exchanging programs during operation.

- Exchanging the entire application program using the KDCPROG tool and the administration command KDCAPPL or an administration program that KDCADMI calls with the KC_CHANGE_APPLICATION operation code.
- Exchanging shared objects, i.e. parts of the application program, using the administration command KDCPROG or an administration program that KDCADMI calls with the KC_MODIFY_OBJECT operation code and KC_LOAD_MODULE object type.

The two methods can be mixed, i.e. an application containing shared objects can also be exchanged as a whole. Both methods are described below.

9.1 Exchanging an application

openUTM offers you the option of exchanging the application program while the application is running. This means that you can modify program units of your application program, create a new version of the application program, and start up this version of the application program without having to terminate the application run, for example.

The applications to be exchanged can consist as well as only of program units that are statically, as of program units that are linked as shared objects.

Please note the following when changing the application program:

- In applications **without** shared objects the changes may not have any effect on the KDCDEF generation or the KDCFILE. This means that you **cannot** use the “exchange applications” function if you want to add new program units to the application program. If new program units are added, then they are not contained in the tables of the KDCFILE.
- In applications **with** shared objects you can add new program units to the application program. These program units must be linked in shared objects, however, that are contained in the configuration of the application. The program units and the associated transaction codes must be entered in the tables of the KDCFILE via the administration.

If there are some program units missing after the exchange that were in the previous application program, then jobs can be submitted to the application for TACs for which there is no program unit any more. These jobs are terminated by openUTM with PEND ER. The transaction codes can be deleted by the UTM administrator from the configuration.

The different versions of your application program and the exchange process are managed by the KDCPROG tool (see [section “The KDCPROG tool”](#)). KDCPROG manages the versions of the application program in a file generation group (FGG=File Generation Group).

If you exchange a program unit which exists as a shared object, the old file is first exchanged by a new one. As a result, the references from the static part of the application program to the shared object are then unresolved. Only when the UTM administrator initiates the program exchange for the application program using the administration command `KDCAPPL PROG=NEW | OLD`, are all unresolved references to the shared object resolved and the new shared object brought into effect in the application program.

i With `KDCAPPL PROG=SAME` you can reload the application program. I.e. if file generations are used, the program of the same file generation is loaded again.

9.1.1 Requirements exchanging an application

The following steps must be carried out in order to exchange an application:

1. Create the file generation group filebase/PROG

In order to exchange a UTM application program during operation, you must administer the different versions of the application program (including the one currently loaded) using the KDCPROG tool. You must use KDCPROG to create the file generation directory PROG in the base directory *filebase* of the application (KDCPROG CREATE function). If you have not created a file generation directory, then KDCAPPL PROG= (or the corresponding call on the administration program interface) reloads the application program *utmwork* from the base directory.

KDCPROG manages the different versions of your application program in the **File Generation Group** (called the FGG in the following). See also [section "File generation group PROG"](#). Each *utmwork* program must thus be stored as a generation in the FGG.

The FGG need only be created once. It remains in existence. In it you can manage several versions of your application program with the aid of KDCPROG.

You can create the FGG before or after you have created the first version of the application program. The application program, i.e. *utmwork*, is created as described in [chapter "Creating the application program"](#).

! CAUTION!

Only one FGG can exist for each application for exchanging the application. Any existing FGG and the versions of the application program it contains are deleted if a new FGG is created with KDCPROG CREATE.

2. Transferring the application program to the FGG

You use KDCPROG TRANSFER to transfer your application program to the FGG PROG (see [section "TRANSFER - transfer utmwork to the FGG"](#)). In this case you must assign the version - i.e. the absolute generation number - 0001 to the application program.

You must also use KDCPROG SWITCH to switch the FGG base to generation number 0001.

You can then start the application, as described in [chapter "Starting a UTM application"](#). openUTM loads *utmwork* from the FGG.

3. Creating more versions of the application program and transferring them to the FGG

Regardless of whether or not your application is started, you can create further versions of your application program. *utmwork* is transferred to the FGG using KDCPROG TRANSFER. A generation number must be assigned to each version of the application program. The generation numbers in the FGG must be assigned in ascending order.

! CAUTION!

If a generation of the application program already exists in the FGG with the generation number you specify when transferring a new generation, this old generation is overwritten without warning.

If these conditions are fulfilled, you can initiate an application exchange at any time and as often as you like. You have the following possibilities to do so:



The administration command `KDCAPPL PROG=...` and the `KDCADMI` operation code `KC_CHANGE_APPLICATION` on the administration program interface. Both possibilities are described in the openUTM manual “Administering Applications”.

If references are made in the following to actions that can be carried out with `KDCAPPL PROG=`, then the same is also true for administration programs that issue `KDCADMI` calls with the operation code `KC_CHANGE_APPLICATION`.

At the start, openUTM loads the version of the application program whose generation number is the base number of the FGG. See also [section “File generation group PROG”](#). If a program is exchanged while the application is running, openUTM sets the base to the version currently loaded. This ensures that the next start is implemented with the version of the application program last loaded.

Notes on exchanging programs a UTM cluster application

In a UTM cluster application, each node application has its own file generation directory `PROG` that you must set up in step 1.

To avoid you having to perform steps 2 and 3 explicitly for each node application, it is recommended that you set up the `PROG` directories in such a way that they are linked (e.g. using `ln -s <filebase1>/PROG <filebase2>/PROG`). This ensures that all node applications always access the same versions of the application program.

9.1.2 File generation group PROG

openUTM manages the different versions of your application, which you use for the exchange, in the file generation group (FGG) PROG. The FGG is created as follows (see also [section “The KDCPROG tool”](#)):

On Unix and Linux systems with the command:

```
utmpath/ex/kdcprog CREATE operand
```

On Windows systems in the DOS command prompt with the command:

```
utmpath\ex\kdcprog CREATE operand
```

In the following table, the central FGG terms are compared with the UTM-specific definitions relating to version management and application program exchange.

FGG terms	Version management for application exchange by openUTM
Generation of the FGG	A version of your application program comprises the <i>utmwork</i> program. The <i>utmwork</i> file forms a <i>generation</i> of the FGG.
Absolute generation number	When transferring a generation (i.e. a new version of the application program) to the FGG, you assign a serial number to the generation, the absolute generation number of the generation. UTM addresses the individual generations by means of their generation numbers.
Name of the generation	The name of a generation is <ul style="list-style-type: none">• on Unix and Linux systems: <i>PROG / absolute_generation_number</i>• on Windows systems: <i>PROG \ absolute_generation_number</i> where $0001 \leq \textit{absolute_generation_number} \leq 9999$. The absolute generation number 0001 must be assigned to the first application program you transfer to the FGG. The name of the generation is thus <i>PROG/0001</i> or <i>PROG\0001</i> .
Ascending generation number	The generation numbers of the subsequent generations must be assigned in ascending order (e.g. 0002, 0003, 0004, etc.).
Base of the FGG	One generation of the FGG is the <i>base</i> of the FGG. When the application starts, UTM always loads the base.
Base number	The absolute generation number of the base is the base number of the FGG. After the FGG is created, 0001 is the base number and the first generation transferred to the FGG is the base of the FGG. While the application is running, UTM always switches the base to the generation currently loaded, i.e. when a program is exchanged UTM switches the base to the generation loaded during exchange.
Relative generation number	Relative to the base, each generation of an FGG is assigned a relative generation number and a relative FGG name.

Relative FGG name	The relative FGG name of a generation is PROG(<i>relative_generation_no.</i>) where: <i>relative_generation_no.</i> = <i>absolute_generation_no.</i> - <i>base_number</i> . The relative generation number is always specified with a sign (+ / -).
Maximum number of generations	When creating the FGG, you define the maximum number of generations that can exist simultaneously in the FGG.
First generation	When the maximum number of generations is reached, UTM executes the next transfer as follows: <ul style="list-style-type: none"> • openUTM creates a new generation with a new generation number • openUTM deletes the generation of the FGG with the lowest generation number, i.e. the first generation.
Last generation	The generation of the FGG with the highest generation number is the last generation of the FGG.

Examples

- When the application starts, the generation with the absolute generation number 0001 is the base of the FGG. While the application is running, the application program is exchanged with KDCAPPL PROG=NEW and generation with generation number 0002 is loaded. This generation is then automatically the base of the FGG. The next time the application starts, openUTM loads this generation (0002). Between two application runs, you can switch the base using the SWITCH function of the KDCPROG tool:
 - The base has the relative generation number +0000.
 - The generation switched to when the application is exchanged with KDCAPPL PROG=NEW, has the relative generation number +0001.
 - The generation switched to with KDCAPPL PROG=OLD has the relative generation number -0001.
- The base number of the FGG is 6. The generation with the absolute generation number 0001 then has the relative FGG name PROG(-5) and the generation with the absolute generation number 0008 has the relative FGG name PROG(+2).
- The maximum number of generations in the FGG is 3. The FGG contains the generations 0001, 0002, 0003 in the PROG directory:
 - 0001 is thus the first and 0003 the last generation.
 - When the generation 0004 is transferred to the FGG, openUTM deletes the first generation 0001. The FGG then contains the three generations 0002, 0003, 0004.
 - The first generation is now the generation with the absolute generation number 0002. This is deleted by openUTM if an additional generation is transferred to the FGG.

9.1.3 Process of exchanging an application

The administrator of the UTM application initiates the exchange of the application program with the command `KDCAPPL PROG=...`, for example. The program exchange then runs separately.

If you specify `KDCAPPL PROG=NEW`, the application program of the generation `PROG(+1)` is loaded; specifying `KDCAPPL PROG=OLD` loads the application program of the generation `PROG(-1)`.

For this reason, it is advisable to define the last generation of the FGG as the base. This generation is then loaded when the application starts. A new generation of the application program is transferred to the FGG with the relative generation number `PROG(+1)` (default setting for `KDCPROG TRANSFER`). When a program is exchanged with `KDCAPPL PROG=NEW`, the new generation of the application program is loaded. `openUTM` automatically switches the base to the generation currently loaded, i.e. the last generation. If you then want to switch back to the application program previously loaded, specify `KDCAPPL PROG=OLD`.

The exchange is implemented for each work process of the application. For each individual work process, the active application program is terminated and the new application program loaded after the current job has been executed. The exchange is not implemented for the next work process until the exchange for this work process has concluded. This means that the application run is not significantly interrupted by the application exchange. In this way, the user is unaware of the application exchange and can continue to work unhindered.

While the application is being exchanged, the process described may result in the situation that jobs are simultaneously processed by individual work processes with the old application program and by other work processes with the new application program. You can prevent this if you issue the administration command `KDCAPPL TASKS=1` to reduce the maximum permitted number of work processes to 1 before the exchange takes place (e.g. with the administration command `KDCAPPL TASKS=1`).

After the application exchange has ended, a UTM message is output to inform the UTM administrator as to whether the exchange was successful or was aborted with errors. The administrator cannot start the next application exchange until this exchange is concluded for all work processes.

9.1.4 The KDCPROG tool

The KDCPROG tool is called as follows:

On Unix and Linux systems you call KDCPROG from a shell with the command:

utmpath/ex/kdcprog function operands

On Windows systems you start a command prompt window and enter the following command:

utmpath\ex\kdcprog function operands

KDCPROG offers the following functions:

Function	Meaning
CREATE	Create the file generation directory PROG for the application exchange.
INFO	Output information on the current state of the FGG.
TRANSFER	Transfer a new version of the application program to a generation of the FGG.
SWITCH	Switch the base number of the FGG.

The operands are described below.

9.1.4.1 CREATE - create a file generation group (FGG)

KDCPROG CREATE creates an FGG for the application exchange. A directory called PROG is created in the *filebase* directory of the application. An existing PROG directory is fully deleted beforehand by KDCPROG.

KDCPROG CREATE *filebase number_entries*

filebase

Name of the directory defined in MAX...,KDCFILE=*filebase* in the KDCDEF generation.

number_entries

Maximum number of generations that can exist simultaneously in the FGG PROG. As soon as *number_entries* exist in the FGG, the first generation of the FGG is deleted when a new generation is transferred to the FGG.

Minimum value: 2

Maximum value: 9999

9.1.4.2 INFO - query the current state of the (FGG)

KDCPROG INFO displays the current state of the FGG. The following data is output:

- current number of entries
- base number of the FGG
- file generation with the lowest generation number (first generation)
- file generation with the highest generation number (last generation)
- list of the file generations contained in the FGG, with absolute and relative names
- list of the existing generations in the PROG directory. The output on Unix and Linux systems corresponds to the output of the `ls -l` command for the PROG directory. A general note regarding the `dir` command is output on Windows systems.

Examples of the output from KDCPROG INFO can be found in [section “Example of exchanging an application”](#).

KDCPROG INFO *filebase*

filebase

Name of the directory defined in MAX...,KDCFILE=*filebase* in the KDCDEF generation.

9.1.4.3 TRANSFER - transfer utmwork to the FGG

KDCPROG TRANSFER transfers *utmwork* from the *filebase* in the FGG.

KDCPROG TRANSFER *filebase generationnumber*

filebase

Name of the directory defined in n MAX...,KDCFILE=*filebase* in the KDCDEF generation.

generationnumber

Number of the generation to which *utmwork* is to be transferred.

Specification of the *generationnumber* is mandatory when transferring the first version of the application program. For *generationnumber* you must specify 0001 (absolute) or +0 (relative). Specification is optional with subsequent transfers. If you do not specify *generationnumber*, then openUTM assumes the value +1.

If you specify a generation for *generationnumber* which already exists in the FGG, this generation is overwritten.

The *generationnumber* can be specified in two ways:

1. Specification of an absolute generation number.

The maximum and minimum value of *generationnumber* depend on the number of FGG entries.

The first generation you transfer to the FGG must always have the absolute generation number 0001.

New generation numbers you assign subsequently must be in ascending order (0002, 0003...). You can also specify generation numbers of generations which already exist in the FGG. These generations are then overwritten.

KDCPROG presets the following limits:

Minimum value: 1

Maximum value: 9999

Note

The specification (current base number - 1) for *generationnumber* specifies the file generation switched to when the application is exchanged with KDCAPPL PROG=OLD.

The specification (current base number + 1) for *generationnumber* specifies the file generation switched to when the application is exchanged with KDCAPPL PROG=NEW.

2. Specification of a relative generation number with sign (+ or -).

The maximum and minimum value of *generationnumber* depend on the current base value and the number of FGG entries. For example, if the last generation (highest absolute generation number) is the base, you must not specify any relative generation numbers greater than +1.

KDCPROG presets the following limits:

Minimum value: - 99

Maximum value: + 99

Note

The specification -1 for *generationnumber* specifies the file generation switched to when the application is exchanged with KDCAPPL PROG=OLD.

The specification +1 for *generationnumber* specifies the file generation switched to when the application is exchanged with KDCAPPL PROG=NEW.

Default value: +1 (relative generation number)

If the default value is specified, the new version of the application program is transferred to the FGG entry switched to with KDCAPPL PROG=NEW.

9.1.4.4 SWITCH - switch the base of the file FGG

With KDCPROG SWITCH, you can switch the base of the FGG when the application is not running. The next time the application starts, the new base of the FGG is then loaded. With KDCPROG SWITCH, therefore, you can execute the functions KDCAPPL PROG=NEW or PROG=OLD between two application runs.

If KDCPROG SWITCH is called while the application is running, the call will be rejected.

KDCPROG SWITCH *filebase basenumber*

filebase

Name of the directory defined in MAX...,KDCFILE=*filebase* in the KDCDEF generation.

basenumber

Specification of the new base generation. For *basenumber*, you can only specify a generation number for which a generation already exists in the FGG.

There are two specification options:

1. Specification of an absolute generation number.

basenumber identifies the new base generation directly.

Minimum value: 0

Maximum value: 9999

2. Specification of a relative generation number with a negative sign.

The base generation is specified relative to the last generation (= generation with the highest generation number). In this case, the value of *basenumber* must always be specified with a leading minus sign (-).

Generation number of base = number of last generation - *basenumber*

Minimum value: -99

Maximum value: -1

Example

The generation with generation number 0010 is the last generation in the FGG.

If KDCPROG SWITCH *filebase -1* is specified, then the file generation with generation number 0009 is the new base of the FGG.

If KDCPROG SWITCH *filebase 0* is specified, the last generation (0010) is the new base of the FGG.

9.1.5 Example of exchanging an application

The sections below present an example application exchange - using the KDCPROG tool.

Step 1

An FGG for application exchange is created that can contain a maximum of three generations of the application program. The first generation of the application program is then transferred to the FGG. This generation is thus the base of the FGG. The current directory (".") is specified for *filebase*.

In Unix and Linux systems:	
Input:	utmpath/ex/kdcprog CREATE . 3
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U376 kdcprog: FGG files for ./PROG created.
Input:	utmpath/ex/kdcprog TRANSFER . +1 utmpath/ex/kdcprog SWITCH . 1
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U383 kdcprog: TRANSFER : /bin/cp ./utmwork ./PROG/0001 U389 kdcprog: TRANSFER successful U388 kdcprog: new base for program FGG ./PROG is 1

On Windows systems:	
Input:	utmpath\ex\kdcprog CREATE . 3
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U376 kdcprog: FGG files for ./PROG created (pid: 348,...)
Input:	utmpath\ex\kdcprog TRANSFER .
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid: 1234,...) U391 kdcprog: TRANSFER for KDCAPPL PROG=NEW initiated U383 kdcprog: TRANSFER : UTMCMD COPY ./utmwork.exe ./PROG/0001 1 file(s) copied. U389 kdcprog: TRANSFER successful
Input:	utmpath\ex\kdcprog SWITCH . 1
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234,...) U388 kdcprog: new base for program FGG ./PROG is 1

Step 2

A new version of the application program is created during the application run. This version is to be transferred to the FGG as the next generation 0002. The relative generation number of this generation is then (+1). This is the default setting for TRANSFER.

In Unix and Linux systems:	
Input:	utmpath/ex/kdcprog TRANSFER .
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U383 kdcprog: TRANSFER : /bin/cp ./utmwork ./PROG/0002 U389 kdcprog: TRANSFER successful

On Windows systems:	
Input:	utmpath\ex\kdcprog TRANSFER .
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U391 kdcprog: TRANSFER for KDCAPPL PROG=NEW initiated U383 kdcprog: TRANSFER : UTMCMD COPY ./utmwork.exe ./PROG/0002 1 file(s) copied. U389 kdcprog: TRANSFER successful

The transferred generation is used if KDCAPPL PROG=NEW is specified.

Step 3

Information on the FGG is requested.

In Unix and Linux systems:	
Input:	utmpath/ex/kdcprog INFO .
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0001 FGG first generation 0001 FGG last generation 0002 File PROG/0001 is PROG(+0000) <= File PROG/0002 is PROG(+0001) The following program files are available: -rwx----- 1 example other 2845876 Apr 22 15:35 ./PROG/0001 -rwx----- 1 example other 2845876 Apr 22 15:37 ./PROG/0002

On Windows systems:	
---------------------	--

Input:	utmpath\ex\kdcprog INFO .
Output:	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0001 FGG first generation 0001 FGG last generation 0002 File PROG/0001 is PROG(+0000) <= File PROG/0002 is PROG(+0001) The following program files are available: kdcprog: type "DIR .\PROG*.EXE" to get full information </pre>

In the output, note that PROG/000x specifies the name of the respective generation.

PROG(+000x) is the relative FGG name. The arrow "<=" points to the base of the FGG; this is the application program generation currently loaded.

Step 4

The UTM administrator exchanges an application. Generation 0002 (alias PROG(+1)) is to be loaded. The administrator thus signs on to the UTM application and enters:

```
KDCAPPL PROG=NEW
```

After the exchange, information is again requested on the FGG.

In Unix and Linux systems:	
Input:	utmpath/ex/kdcprog INFO .
Output:	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0002 FGG first generation 0001 FGG last generation 0002 File PROG/0001 is PROG(-0001) File PROG/0002 is PROG(+0000) <= The following program files are available: -rwx----- 1 example other 2845876 Apr 22 15:35 ./PROG/0001 -rwx----- 1 example other 2845876 Apr 22 15:37 ./PROG/0002 </pre>

On Windows systems:	
Input:	utmpath\ex\kdcprog INFO .

Output:	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0002 FGG first generation 0001 FGG last generation 0002 File PROG/0001 is PROG(-0001) File PROG/0002 is PROG(+0000) <= The following program files are available: kdcprog: type "DIR .\PROG*.EXE" to get full information </pre>
----------------	---

The output indicates that openUTM has changed the base. The base is now the generation with generation number 0002, which was loaded when the application was exchanged. Generation 0001 is used if KDCAPPL PROG=OLD is entered. No program is available for KDCAPPL PROG=NEW.

Step 5

Another version of the application program is transferred to the FGG. This means that a new version of the application program is available for another application exchange with KDCAPPL PROG=NEW.

Information on the FGG is again requested after the transfer.

In Unix and Linux systems:	
Input:	<pre> utmpath/ex/kdcprog TRANSFER . </pre>
Output:	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U383 kdcprog: TRANSFER : /bin/cp ./utmwork ./PROG/0003 U389 kdcprog: TRANSFER successful </pre>
Input	<pre> utmpath/ex/kdcprog INFO . </pre>
Output	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0002 FGG first generation 0001 FGG last generation 0003 File PROG/0001 is PROG(-0001) File PROG/0002 is PROG(+0000) <= File PROG/0003 is PROG(+0001) The following program files are available: -rwx----- 1 example other 2845876 Apr 22 15:35 ./PROG/0001 -rwx----- 1 example other 2845876 Apr 22 15:37 ./PROG/0002 -rwx----- 1 example other 2845876 Apr 22 15:43 ./PROG/0003 </pre>

On Windows systems:

Input:	utmpath\ex\kdcprog TRANSFER .
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234,...) U391 kdcprog: TRANSFER for KDCAPPL PROG=NEW initiated U383 kdcprog: TRANSFER : UTMCMD COPY ./utmwork.exe ./PROG/0003 1 file(s) copied. U389 kdcprog: TRANSFER successful
Input	utmpath\ex\kdcprog INFO .
Output	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0002 FGG first generation 0001 FGG last generation 0003 File PROG/0001 is PROG(-0001) File PROG/0002 is PROG(+0000) <= File PROG/0003 is PROG(+0001) The following program files are available: kdcprog: type "DIR .\PROG*.EXE" to get full information

Now, a program is available for exchanging an application with KDCAPPL PROG=OLD and a program is available for exchanging an application with KDCAPPL PROG=NEW.

Step 6

Another version of the application program is transferred to the FGG and KDCPROG INFO is called.

In Unix and Linux systems:	
Input:	utmpath/ex/kdcprog TRANSFER .
Output:	U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U383 kdcprog: TRANSFER : /bin/cp ./utmwork ./PROG/0004 U389 kdcprog: TRANSFER successful
Input	utmpath/ex/kdcprog INFO .

Output:	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0002 FGG first generation 0002 FGG last generation 0004 File PROG/0002 is PROG(+0000) <= File PROG/0003 is PROG(+0001) File PROG/0004 is PROG(+0002) The following program files are available: -rwx----- 1 example other 2845876 Apr 22 15:37 ./PROG/0002 -rwx----- 1 example other 2845876 Apr 22 15:43 ./PROG/0003 -rwx----- 1 example other 2845876 Apr 22 15:59 ./PROG/0004 </pre>
----------------	---

On Windows systems:

Input:	utmpath\ex\kdcprog TRANSFER .
Output:	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid: 1234,...) U391 kdcprog: TRANSFER for KDCAPPL PROG=NEW initiated U383 kdcprog: TRANSFER : UTMCMD COPY ./utmwork.exe ./PROG/0004 1 file(s) copied. U389 kdcprog: TRANSFER successful </pre>
Input	utmpath\ex\kdcprog INFO .
Output	<pre> U181 Program kdcprog V07.0A00 is started on <system + bit mode> (pid:1234, ...) U378 INFO for FGG ./PROG FGG maximum number of versions 3 FGG base 0002 FGG first generation 0002 FGG last generation 0004 File PROG/0002 is PROG(+0000) <= File PROG/0003 is PROG(+0001) File PROG/0004 is PROG(+0002) The following program files are available: kdcprog: type "DIR .\PROG*.EXE" to get full information </pre>

The generation 0001 has been deleted because a maximum of three generations can be contained in the FGG.

9.2 Exchanging shared objects

With the “exchange shared objects” function you can exchange individual parts of the application program during operation. These application parts must be created as shared objects and linked dynamically to the application. In this case, you must carry out certain steps when compiling, linking and generating.

Shared objects are implemented using DLLs on Windows systems. More details can be found in the [section “Creating application programs as DLLs \(Windows systems\)”](#).



You can exchange shared objects with the administration command KDCPROG or with a separate administration program that calls KDCADMI with the operation code KC_MODIFY_OBJECT and object type KC_LOAD_MODULE. Both possibilities are described in the openUTM manual “Administering Applications”.

You can also exchange shared objects with the administration tools WinAdmin/WebAdmin.

If a reference is made in the following to an actions that can be carried out with the KDCPROG command, then this is also true for administration programs that issue KDCADMI calls with the operation code KC_MODIFY_OBJECT and object type KC_LOAD_MODULE.

An application with shared objects can also be exchanged as a whole.

9.2.1 Providing and generating shared objects

A shared object in C must always be compiled on Unix and Linux systems such that the respective runtime system is also linked.

More details on compiling applications on Windows systems can be found on "[Compiling and linking the application \(Windows systems\)](#)".

Version concept of shared objects

Shared objects can be created with or without versions.

- Without versions
If you want to provide a shared object without a version, you must supply precisely one file with the shared object. When exchanging using the administration command KDCPROG, it is sufficient to specify the name of the file. Shared objects without versions can only be loaded dynamically at the start of the application.
- With versions
If a shared object is to be available in several versions, you must first create a directory and then copy the individual versions of the shared object into this directory. You can add as many versions as you wish. When exchanging, specify both the directory name of the shared object and the version name.
On Windows systems, shared objects should always be created **with** versions.

Generating shared objects

Each shared object must be generated with the KDCDEF statement SHARED OBJECT (see openUTM manual "Generating Applications"). Specify the following:

- The name of the shared object. If the shared object has no versions, specify the name of the file under which it is stored. If the shared object has versions, specify the name of the directory containing the versions.
On Windows systems, the name of the shared object must have the extension `.dll`.
Only one version can be generated for each shared object. The version can be changed using UTM administration functions.
- The file name of the respective version, if the shared object has a version (VERSION operand).
On Windows systems, it is necessary to specify the version.
- The path name under which the shared object can be found (DIRECTORY operand).
On Windows systems, you should always specify the complete path because the PATH and LD_LIBRARY_PATH environment variables are not evaluated for shared objects.
- Whether the shared object is to be loaded at the start of the application (LOAD-MODE=STARTUP) or with the first call (LOAD-MODE=ONCALL).

The name of the shared object must be specified in the PROGRAM statement belonging to the program unit (SHARED-OBJECT operand, see also the examples on "[Examples of exchanging shared objects](#)").

9.2.2 Start of the application

When the application starts, openUTM loads all the shared objects generated with LOAD=STARTUP in the sequence in which the SHARED-OBJECT statements were specified. Shared objects with LOAD-MODE=ONCALL are not loaded until the first call is issued.

If a shared object cannot be loaded, the startup is continued anyway. If this shared object is called at a later stage, the result is a BADTACS or a PEND ER.

If the event exits START, SHUT or INPUT, or the event services MSGTAC or SIGNON, or the administration program unit KDCADM cannot be loaded, then the startup is terminated with an error message.

9.2.3 The exchange process

The openUTM administrator must initiate the exchange of a shared object with the administration command KDCPROG, for example. In this case, the event exits START and SHUT are not executed unless the application program is terminated and loaded dynamically by the exchange.

If you exchange a shared object that was generated with versions, you must specify the directory name and the version name. In the case of shared objects without versions, specify the name of the shared object itself; a version specification will be ignored.

The exchange process runs differently depending on the time (STARTUP or ONCALL) at which the shared object was loaded.

9.2.3.1 Exchanging shared objects with LOAD-MODE=STARTUP

When exchanging application parts generated with LOAD-MODE=STARTUP, the work process continues running. The shared object is unloaded and the specified version is loaded dynamically.

This program exchange can be executed simultaneously by several work processes of an application. During the program exchange, different states of the application program are loaded in the work processes of the UTM application. Each work process of the application implements the requested program exchange when it has finished processing the current job. A UTM message is output to indicate that the program exchange process is complete.

No further program can be exchanged until this program exchange is concluded. Another KDCPROG call will be rejected by openUTM.

9.2.3.2 Exchanging shared objects with LOAD-MODE=ONCALL

Shared objects generated with LOAD-MODE=ONCALL can only be exchanged if they were generated **with** versions.

If you exchange this type of shared object, only the version identifier to be loaded dynamically for the respective shared object is entered in the openUTM tables when the KDCPROG administration command is processed.

The new version is not loaded by each work process of the application until the next time this work process calls a program unit contained in this shared object. The program exchange can be implemented simultaneously by several work processes of an application. Until the requested program exchange has been implemented by all work processes of the openUTM application, different states of the application program are loaded in the individual work processes. However, it is ensured that each work process implements the requested exchange before another program unit is activated which is contained in the shared object to be exchanged.

The exchange of a shared object generated with ONCALL does not have a blocking effect on subsequent commands for program exchange. Immediately after processing the KDCPROG command, the administrator can therefore initiate another program exchange with another KDCPROG command.

If the version identifiers of the new and old shared object are the same, no program exchange is implemented.

9.2.4 Examples of exchanging shared objects

Example 1 (Unix and Linux systems)

On a Unix or Linux system, an individual module called INCTAX is to be exchanged dynamically on a particular date. In this case, you must carry out the following steps:

1. Create and compile the module using the options required for shared objects.
2. Compile the module with the switches required for shared objects. Transfer the module as a shared object without version under the name INCTAX to the directory containing the user-specific programs. In the example the placeholder *so-lib* is specified for the directory. This directory could be */usr/proglib*, for example.
3. Generate the module as a shared object with the following KDCDEF statements:

```
SHARED-OBJECT INCTAX , DIRECTORY=so-lib , LOAD-MODE=STARTUP  
PROGRAM . . . . , SHARED-OBJECT=INCTAX
```

The shared object is thus loaded when the application starts (mandatory for shared objects without version).

4. Link the work process, whereby you must specify the dynamic library with the shared object INCTAX.
5. Start the application as usual.
6. Modify the module and save it before the effective date in the file:

```
so-lib/INCTAX
```

7. Enter the following administration command:

```
KDCPROG SHARED-OBJECT=INCTAX
```

The shared object will be exchanged in the individual work processes as soon as these processes have processed the current job.

Example 2

On a Unix or Windows system, a shared object called MNTHBALANCE is to be available in 12 versions (BIL01,... BIL12) and is to be exchanged at the end of each month. In each case, it should not be loaded until it is called for the first time. Carry out the following steps:

1. Create and compile the module using the options required for shared objects (for Windows systems see ["Creating application programs as DLLs \(Windows systems\)"](#)).
2. Compile the module with the switches required for shared objects. In the directory with the user-specific programs (*so-lib* in the example), create the directory MNTHBALANCE and into this directory copy at least the version required for the first application run.
3. Generate each version of the shared object with the following KDCDEF statement:

```
SHARED-OBJECT MNTHBALANCE  
                , DIRECTORY=so-lib  
                , VERSION=BILXX      (XX=01 , . . . , 12)  
                , LOAD-MODE=ONCALL
```

The shared object is not loaded until the program unit is called.

Enter the following PROGRAM statement for the shared object:

```
PROGRAM . . . . , SHARED-OBJECT=MNTHBALANCE
```

4. In Unix and Linux systems you link the work process with the dynamic library by specifying the dynamic library with the shared objects.

There are no special cases on Windows systems in this case.

5. Start the application as usual.

6. On July 1, for example, enter the following administration command:

```
KDCPROG SHARED-OBJECT=MNTHBALANCE,VERSION=BIL07
```

Please make sure that this version exists in the directory at the specified time.

The shared object is not exchanged in the individual work processes until the corresponding program unit is called for the first time.

9.2.5 Exchanging an application with shared objects

The entire application can be exchanged with the administration command `KDCAPPL PROG=NEW`. The exchange can be done via the PROG file generation directory that you must prepare with the `KDCPROG` tool (see [section "Requirements for exchanging an application"](#)) or UTM loads the application program `utmwork` directly from the base directory.

In both cases, each work process of the application is unloaded in succession and then loaded dynamically when the entire application is being exchanged. In the dynamic loading process, the new versions of the shared objects are loaded. To minimize the interruption to the operation of the application, the exchange is only performed by one application work process at a time.

9.2.6 Adding programs dynamically

Amongst other things, dynamic administration allows programs to be generated while the application is running. For more details on dynamic administration, see the openUTM manual “Administering Applications”.

These programs must be loaded before they can be called. To this end, the program must be linked with the assigned shared object and must be made available with a new version in the directory specified in the SHARED-OBJECT statement when generating.

The administrator must then exchange this shared object using the KDCPROG command or by calling the program.

The UTM administrator must enter the new program units and the corresponding transaction codes dynamically into the KDCFILE tables.

10 Fault tolerance of openUTM

Fault tolerance in this context means that a UTM application can still remain operational when errors occur in individual program units that force openUTM to abort a transaction. openUTM then ensures that the application program is terminated and reloaded so that the error does not spread any further and have a negative effect on other users of the application and their data.

With regard to the error behavior of openUTM, a distinction is made between:

- Internal UTM errors and errors in the system environment

These errors result in an abnormal termination of the application, just like the administration command `KDCSHUT KILL` or when issuing a `KDCADMI` call with operation code `KC_SHUTDOWN` and subcode `KC_KILL`. openUTM creates a UTM dump for each process of the application. The UTM dump is edited using the UTM tool `KDCDUMP`. A description of this procedure can be found in the openUTM manual "Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems".

- In the event of serious errors in the dialog terminal process, the dialog terminal process terminates and writes a core dump under the current directory. During this sign-on run, it is not possible to sign on again from the assigned terminal. With minor errors, the dialog terminal process signs off properly from the application.
- A printer process behaves similarly to a dialog terminal process when errors occur. The printer process can, if necessary, be restarted using an administration command.
- If errors occur in the timer process, the application is terminated abnormally as soon as a job is sent to the timer process from the work processes.

- Errors in the application program

These are errors in program units. They can be divided into two groups:

- errors that lead to the reloading of the application
- errors that may permit the program to continue.

10.1 Errors detected by openUTM

A program unit is terminated abnormally by openUTM in the following situations:

- A PEND ER or FR was programmed.
- A UTM call supplied a KDCS return code $\geq 70Z$. In this case, openUTM internally sets PEND ER.

In both situations, openUTM aborts the service. If a PEND FR was programmed, then openUTM does not take any other action.

If the service was terminated by a PEND ER (in a program or internally), then openUTM creates a UTM dump with REASON=PENDER that only conveys the data of the KDCROOT. The affected work process is then terminated. The main process subsequently starts a new work process, which dynamically loads the application program. This brings the static data areas to a new state and avoids follow-up errors due to the overwriting of data.

10.2 Signal handling in openUTM

When a signal occurs, the following actions are possible:

- The signal is ignored (see table).
- If the signal occurs while program components created by the user are running: A PENDING with KCRCC=70Z and KCRCDC=XTxx is called (xx is the signal number). The affected service or work process is terminated.
Exception: If a registered user signal routine is existing on a Unix or Linux system, then this routine is called instead of that. Please refer to the openUTM manual „Programming Applications with KDCS” for details.
- If the signal occurs while the UTM system components are running:
The application is terminated abnormally with REASON=SIGxxx (xxx is the signal number).

The table below shows the reaction of a work process to the individual signals. Detailed information on signals can be found in the C header file for signals (signal.h).

Signal received	Reaction
SIGHUP	ignored
SIGINT	ignored
SIGQUIT	ignored
SIGILL	PENDING/TRMA ¹
SIGTRAP	ignored
SIGABRT	ignored
SIGEMT	ignored
SIGFPE	PENDING/TRMA ¹
SIGBUS	PENDING/TRMA ¹
SIGSEGV	PENDING/TRMA ¹
SIGSYS	PENDING/TRMA ¹
SIGPIPE	ignored
SIGALRM	PENDING ² / ignored
SIGTERM	ignored
SIGUSR1	ignored
SIGUSR2	ignored
SIGCHLD	ignored

SIGPWR	PEND ER/TRMA ¹
SIGWINCH	ignored
SIGURG	ignored
SIGIO	ignored
SIGTSTP	ignored
SIGCONT	ignored
SIGTTIN	ignored
SIGTTOU	ignored
SIGVTALRM	ignored
SIGPROF	ignored
SIGXCPU	PEND ER/TRMA ¹
SIGXFSZ	PEND ER/TRMA ¹

¹TRMA stands for Term Application (= terminate application)

²PEND ER if timer is activated (KDCDEF generation TAC RTIME=<rtime>)

If the termination handling caused by the signal is interrupted by the arrival of another signal, this signal is not intercepted by openUTM. In this case, the operating system takes over the handling of the signal interrupt.

! Signals in user signal routine

If another signal occurs during the user signal routine, the standard handling of the operating system is activated for this signal. This leads to an abnormal termination of the UTM application.

10.3 Termination of application by system crash / shutdown

A system crash or a shutdown results in an abnormal termination of the application, whereby no UTM dump is created. All processes of the application are terminated by the operating system. Before the application is restarted in such situations, the UTM tool KDCREM must be called. See also [section "The KDCREM tool"](#).

11 Accounting

openUTM provides accounting functions that enable the user of a UTM application to calculate the resources utilized by the users of a UTM application.

The accounting functions that the corresponding operating system provides can only record the resource utilization and performance of a UTM application as a whole. However, if you want to be able to assign the computer resources used to individual users and charge the individuals accordingly, then the following must be taken into account for UTM accounting:

- The users of a UTM application are represented by the user IDs defined in the UTM generation and not by the user IDs of the operating system. You must therefore be able to assign the resources used by a user to individual UTM user IDs.
- A group of homogenous processes is active in a UTM application. Every process handles a series of jobs in succession for various users. The resources used within a process must therefore be determined for each service called (i.e. for individual program unit runs).
- The time conditions of OLTP operation require that the services be recorded in such a way that the performance of the application is not impeded.

UTM accounting therefore records the utilization of resources by the individual program units. This means that the resource utilization can be assigned to the transaction code (TAC) of the respective program unit and therefore to the UTM user who started the corresponding service.

In addition to the utilization of resources determined by UTM accounting, there is also a basic resource requirement that arises when a UTM application is running but which cannot be assigned directly to a user. These are:

- Disk space assignment for KDCFILE, SYSLOG, and USLOG files
- CPU utilization for
 - starting and terminating UTM processes
 - handling connections for terminals
 - LPUT handling (transfer to USLOG file)
 - processing printer output (Unix and Linux systems)

If the usage of these resources is to be taken into account, then you must charge these services at a flat rate to the users.

11.1 Definition of terms

This section provides a more detailed explanation of some of the terms that are relevant to UTM accounting.

Users in the sense of UTM accounting

The user of a UTM application for whom an account is to be created, is represented by the UTM user ID.

openUTM assigns the utilized resources to the LTERM partners as an alternative in UTM applications without real user IDs. The LTERM name of the connection user ID (TS applications and UPIC clients), the LU6 session name (LU6 partners) or the OSI association name (OSI TP partner) is used for applications or clients that have not explicitly signed on with a user ID.

In UTM applications without user IDs, openUTM assigns the resources used by terminals, UPIC clients or TS applications to the LTERM partners instead.

Accounting file

All information that the UTM accounting collects for the user-specific accounting of resources used is written by openUTM in the accounting file.

The accounting file is maintained according to the application, and administered by the administrator of the UTM application; for more details see [section "Evaluation"](#).

Resources

This includes the following services:

- technical DP services, particularly CPU utilization
- calling a particular program (program charge)

Calculation phase

The calculation phase is used as a starting point for the utilization of the accounting procedure.

In the calculation phase, openUTM determines the utilization of each resource for each program unit called and writes the values in the BS2000 accounting file as a calculation record. See [section "Calculation phase"](#) for more detailed information.

Calculation record

A calculation record is a record which openUTM writes in the BS2000 accounting file for each program unit run in the calculation phase. The accounting record type is UTMK. The data fields of the calculation record UTMK are described in the Appendix on ["Structure of a calculation record"](#).

Weight

A weight (factor) can be defined for each resource. This weight specifies how the resource is to be evaluated compared with other resources. The utilization of a resource is then introduced into the accounting procedure as the product "weight * resource utilization". The weights for the individual resources are entered in the KDCDEF generation in ACCOUNT, see [section "Determining the variant of the accounting procedure"](#).

Accounting phase

openUTM determines the resource utilization for each program unit. When the program unit terminates, openUTM calculates the sum of utilization values based on the weights and the generated fixed prices.

The following resources are taken into account:

- CPU utilization
- generated output jobs for printers (Unix and Linux systems)
- fixed price for calling a program unit

The result is a number of derived accounting units that are added to the user-specific accounting unit counter.

openUTM only then writes a record with the contents of this counter in the accounting file

- when the user signs off and is not signed on again to the UTM application via any other connection,
- when the application is terminated normally,
- or when a particular (generatable) maximum value is exceeded. You specify this maximum value in the KDCDEF generation with `ACCOUNT ...,MAXUNIT= .`

You must incorporate the weights in the UTM generation of the application before the start of the accounting phase. You can choose between the following:

- fixed-price accounting
- utilization-oriented accounting
- combination of both variants

You will find a detailed description of the accounting phase in [section "Accounting phase"](#).

The accounting phase of UTM accounting can be enabled and disabled while the UTM application is running.

Accounting record

An accounting record is a record which openUTM writes to the accounting file in the accounting phase. The accounting record type is UTMA.

The data fields of the accounting record UTMA are described in the Appendix on "[Structure of an accounting record](#)"

.

Accounting units

Accounting units are the product of the utilization and weight of the respective resource. Only accounting units are counted in the UTM accounting facility.

Accounting unit counter

In a UTM application, openUTM keeps an accounting unit counter for each user and thereby accumulates the utilization of accounting units per user.

Fixed-price accounting

With this variant of the accounting function, a constant number of accounting units is calculated for a program unit run. This number is assigned to the transaction code when the application is generated. The weights of other resources are zero. In this manner you can also offer free services, e.g. informational functions.

Utilization-oriented accounting

With this variant of the accounting function, the current utilization of resources is calculated for a program unit run. The utilization values for the resource are weighted according to the generated weights. No fixed price is charged for calling program units.

11.2 Accounting phases

The following steps are required to execute accounting in UTM applications:

- calculation phase
- determination of the accounting procedure
- accounting phase
- evaluation

11.2.1 Calculation phase

The calculation phase provides approximate values that you can use to determine the weights and fixed prices for the utilization of a service. openUTM determines the resource utilization for each program unit run, creates a calculation record of type UTMK at the end of the program run and writes this record in the accounting file.

The calculation phase can also be enabled or disabled at any time via the UTM administration during live operation to check the generated weights and possibly to update them when regenerating, for example.

You should note, however, that openUTM writes a record in the accounting file after every program unit run when the calculation phase is activated. This has a negative impact on the performance of the application.

Activating the calculation phase

The calculation phase can be activated during KDCDEF generation or by administration, see openUTM manual “Generating Applications” and openUTM manual “Administering Applications”:

- KDCDEF statement ACCOUNT ACC=CALC
- or via UTM administration:
 - using the KDCAPPL CALC=ON command
 - or using WinAdmin/WebAdmin
 - or using the KDCADMI program call KC_MODIFY_OBJECT with obj_type=KC_DIAG_AND_ACCOUNT_PAR

Deactivating the calculation phase

The calculation phase can only be deactivated by UTM administration:

- using the KDCAPPL CALC=OFF command
- or using WinAdmin/WebAdmin
- or using the KDCADMI program call KC_MODIFY_OBJECT with obj_type=KC_DIAG_AND_ACCOUNT_PAR

Data of a calculation record

A calculation record contains the following data:

- name of the UTM application
- transaction code (TAC) of the program unit
- CPU utilization in the UTM task (msec)
- length of the input message in bytes
- length of the output message in bytes
- number of output jobs to printers (Unix and Linux systems)
- accounting units for LTAC calls
- UTM users that have called the service
- name of the LTERM partner through which the user is signed on
- real time of the program unit run (msec)

Output messages that are intended for a follow-up program unit (e.g. after PEND ER) are also counted.

11.2.2 Determining the variant of the accounting procedure

You must first determine if you want to use fixed prices, the utilization or a combination of these two variants for accounting purposes. Your decision depends on if you want to offer certain services of the application at fixed prices or if you want to charge for the actual resource utilization.

Fixed-price accounting

In fixed-price accounting, a program unit run costs a constant number of accounting units. These values are based on the values determined in the calculation phase. This makes fixed-price accounting the simplest solution.

You specify the number of accounting units in the KDCDEF generation in the TAC statement in the TACUNIT operand, see the openUTM manual “Generating Applications”.

```
TAC tacname , PROGRAM=programe , TACUNIT=number_of_accounting_units
```

The value specified in TACUNIT is added to the user-specific accounting unit counter for every transaction code called by the user.

You can also provide some services (e.g. informational functions) free of charge when using fixed-price accounting. You must generate the corresponding transaction codes as follows to do this:

```
TAC ... TACUNIT=0
```

With distributed processing, the same applies to the LTAC statement and the LTACUNIT operand, see [section “Accounting with distributed processing”](#).

You must set the weights for the resources to 0 (default value) in the KDCDEF statement ACCOUNT when using fixed-price accounting.

Utilization-based accounting

In this variant the user is charged for the utilization of resources that are determined in the current accounting phase. You must specify weights for the individual resources. A weight is a factor that is multiplied with the number of units used. You can use the utilization data that you received in the calculation phase to help you choose the weights.

The weights are defined for each application in the KDCDEF statement ACCOUNT, i.e. they are valid for all program unit runs.

The determination of the weights is inevitably subjective and depends on the installation environment. You can assign weights to the following resources:

- CPU utilization (ACCOUNT operand CPUUNIT)
- I/O to background memory (ACCOUNT operand IOUNIT)
- printer output (ACCOUNT operand OUTUNIT) (Unix and Linux systems)



More details can be found in the openUTM manual “Generating Applications”.

Example for the KDCDEF generation of this variant

```
ACCOUNT ACC=ON, CPUUNIT=15, IOUNIT=5, OUTUNIT=20
```

```
TAC tacname, PROGRAM=programe, TACUNIT=0
```

```
TAC . . . .
```

The following sum is then added to the accounting unit counter of the user for each transaction code call:

$15 * \text{CPU utilization} + 5 * \text{I/O utilization} + 20 * \text{printer output utilization}$

Combination of fixed-price and utilization-based accounting

You can also combine the two variants above for your accounting purposes by specifying a certain fixed price for calling a transaction code and then also charging for the utilization of resources (e.g. the CPU utilization).

The following sum is created and added to the accounting unit counter of the user in the accounting phase when a transaction code is called:

```
TACUNIT (fixed price for calling a program unit)  
+ CPUUNIT * CPU utilization + IOUNIT* I/O utilization  
+ OUTUNIT * printer output utilization
```

Example for the KDCDEF generation of this variant

```
ACCOUNT ACC=ON, CPUUNIT=15
```

```
TAC tacnam1, PROGRAM=programe1, TACUNIT=1
```

```
TAC tacnam2, PROGRAM=programe2, TACUNIT=2
```

```
:
```

```
:
```

11.2.3 Accounting phase

In the accounting phase, openUTM determines the resources utilized per program unit run, calculates a weighted total from this figure and from the generated weights and fixed prices. openUTM then adds this result to the accounting unit counter of the UTM user. The value of this counter is contained in the accounting record which openUTM writes in the accounting file.

openUTM always writes an accounting record when a certain number of accounting units have been accumulated for the user, or when the user signs off and is not signed on to the UTM application via any other connection. The number of accounting units for which openUTM writes an accounting record is specified in the KDCDEF generation in `ACCOUNT MAXUNIT=`. You must note the following:

- You should not select a value for MAXUNIT that is too small because writing accounting records too often could affect the performance of the application negatively.
- You should not select a value for MAXUNIT that is too large because the accounting units that have not yet been written to the accounting file could be lost when the application crashes (accounting is not subject to transaction management).

After the accounting record has been written to the accounting file, the accounting unit counter and the counter for the number of TACs called are reset to zero.

Activating the accounting phase

With the KDCDEF control statement `ACCOUNT ACC=ON`, accounting is also activated for the UTM application in the UTM generation.

The accounting phase can also be activated and deactivated during live operation by the UTM administration.

- using the `KDCAPPL ACCOUNT=ON` command
- or using WinAdmin/WebAdmin
- or using the KDCADMI program call `KC_MODIFY_OBJECT` with `obj_type=KC_DIAG_AND_ACCOUNT_PAR`

Deactivating the accounting phase

The accounting phase can only be deactivated by administration:

- using the `KDCAPPL ACCOUNT=OFF` command
- or using WinAdmin/WebAdmin
- or using the KDCADMI program call `KC_MODIFY_OBJECT` with `obj_type=KC_DIAG_AND_ACCOUNT_PAR`

Data of the accounting record

The accounting record is of record type UTMA. The accounting record contains the following data:

- name of the UTM application
- UTM user ID
- time the user signs on via the current connection
- value of the accounting unit counter
- number of TACs called with `TACUNIT > 0` since the sign-on or since the last record was written

You can also collect calculation data while the accounting phase is running. This allows you to check the weights at any time.

11.2.4 Evaluation

The results of the accounting phase are the accounting records in the accounting file. openUTM creates the accounting file in the subdirectory ACCNT of the base directory *filebase* of the UTM application. The accounting file is named *0001.pid* where *pid* is the process ID of the logging process of the UTM application in which the *utmlog* program is running.

openUTM writes the accounting records and the calculation records in the file *0001.pid*.

After deactivating and then reactivating the accounting phase, openUTM continues to write the records to the same file. The calculation and accounting records collected before deactivating are not overwritten.

You can evaluate the file yourself.



The structure of the accounting records is described in the Appendix on "[Structure of an accounting record](#)".

11.2.5 Error situations

If accounting cannot write an accounting record due to an error, e.g. because there is not enough space on the disk, openUTM generates message K079 and terminates the calculation and/or accounting phase. An insert of message K079 contains the cause of the error. The application continues execution.

After the error has been corrected, the calculation and/or accounting phase can be reactivated again by the UTM administration (e.g. using the administration command KDCAPPL).

11.3 Accounting with distributed processing

During distributed processing, every participating application can, in principle, start services in other applications. Accounting in distributed processing is primarily of use when the roles are unevenly distributed, i.e. one application acts entirely as the job submitter and other applications assume the job receiver roles. Consequently, in this section, the applications are referred to as **job-submitting applications** and **job-receiving applications**.

The job-submitter application (job submitter) uses services provided by program units in remote partner applications (job receivers). In this case, the job-submitting application can be charged with the incurred resource utilization as a fixed price. Accounting units are assigned as a fixed price to the LTACs in the job-submitting application to do this. LTACs are the transaction codes that are defined in the job-submitting application for a service in a job-receiving application.



More details can be found in the openUTM manual “Generating Applications”, LTAC statement, LTACUNIT operand.

Calculation phase (determining the fixed price)

The average resource utilization of the program units that provided services for the job-submitting application is determined in the calculation phase in the **job-receiving application**. You can specify fixed prices based on the utilization values determined that will be charged to the users of LTACs in the job-submitting application.

openUTM counts the accounting units used in the LTAC calls in a field of the calculation record in the **job-submitting application**.

Accounting phase

In the **job-receiving application**, all utilization values that are incurred while processing jobs for a job-submitting application are assigned as follows:

- With LU6.1, to the sessions (LSES) to the job submitter
- With OSI TP, to the associations (OSI-LPAP ... ,ASSOCIATION-NAME=), if the OSI TP-job submitter did not sign on under a real user ID

The total for the services provided is therefore charged to the job-submitting application. The resources used by the individual users of the job-submitting application cannot be determined.

In the **job-submitting application**, openUTM adds the number of accounting units specified in the LTAC statement in the KDCDEF generation when an LTAC is called to the accounting unit counter of the user of the local application.

11.4 Restrictions

Please note the following when using UTM accounting:

- Transaction logging is not implemented when writing accounting information; this means that accounting units may be lost if an application crashes. The maximum value per user can be limited in the UTM generation.
- For applications with distributed processing, each LTAC call is counted in the calculation phase. No account is taken of whether or not a session could be opened following PEND processing.
- The recording of resource utilization begins before a program unit starts and ends with the processing of the PEND call. The remaining processing power (basic utilization) of the UTM processes is not charged to the users.
- Rolling back a transaction has the following effects: All values except for CPU are reset. Since openUTM accumulates the utilization values in the PEND processing, a rollback action can only reset utilization values if they originate in the current program unit run.
- If only asynchronous jobs have been processed for the user since the last application start, the sign-on time to the application is shown as zero in the accounting record.
- For the event exit VORGANG, the resource utilization is only recorded at the start of the service.
- For the event service BADTACS, the program unit weight cannot be taken into account in the accounting phase.

12 Checking performance with openSM2 and KDCMON

The performance of a UTM application is influenced by various factors. The determining factors lie on the one hand in the system environment of a UTM application (configuration of the working memory, performance capabilities of peripherals) and on the other hand in the UTM application itself (configuration of the application and structure of the program units). Performance checks should be carried out at regular intervals while an application is running, in order to detect performance bottlenecks at an early stage. The following tools are available for checking the performance of UTM applications:

- Software Monitor openSM2
- UTM event monitor KDCMON with the evaluation tool KDCEVAL
- information services of UTM administration

openSM2 software monitor

You can monitor the performance of the UTM application using the openSM2 software monitor. For details, refer to [section "Monitoring with openSM2"](#).

UTM event monitor KDCMON

The UTM event monitor KDCMON is provided for UTM users. KDCMON is a function integrated in openUTM and records information on the runtime characteristics of UTM applications and user program units. If performance bottlenecks are detected, then you can collect data using **KDCMON**. You evaluate the data collected with the KDCEVAL tool. You can then carry out a detailed analysis based on this evaluation. See "[Evaluation lists](#)".

KDCMON is therefore an important tool for assessing the performance of a UTM application. KDCMON can be used to produce detailed performance evaluations when measurements using the UTM administration point to a performance bottleneck.

Information services in the UTM administration

Some information on diagnosing performance bottlenecks can also be queried using the **UTM administration information services**, e.g. via the KDCINF administration command or via the graphical administration tools WinAdmin/WebAdmin. The KDCINF STATISTICS command provides data on the utilization of individual selected components of your UTM application (e.g. clients). The KDCSINF STATISTICS command also allows you to obtain general statistical information on the utilization of the entire application and obtain statistics for performance control as well as for assessing the performance of your UTM application during operation, for example application load, page pool utilization, number of users currently signed on, number of dialog or asynchronous transactions performed per second, open dialog and asynchronous services etc.

The KDCINF PAGEPOOL command supplies further, more detailed data on the current utilization of the page pool.

For detailed information on KDCINF, see the openUTM manual "Administering Applications".

If you administer the UTM application with the WinAdmin or WebAdmin graphical administration workstation, then you can also display the statistical data graphically.

12.1 Monitoring with openSM2

The openSM2 software monitor provides comprehensive monitoring data for monitoring the performance of server systems and storage systems. As of openSM2 V9.0, support is also provided for acquiring data specific to UTM applications.

You should make use of the functionality offered by openSM2 to monitor the total system load and the behavior of a UTM application in particular and to uncover performance bottlenecks.

The openSM2 monitoring data does not, however, permit any conclusions to be drawn about individual objects of the UTM application, such as program units. Rather, they show the behavior of the entire application, for instance average values for the transaction rate, the throughput and the processing time.

The conditions listed below must be met to allow openUTM to deliver data to openSM2 and openSM2 to acquire, store and prepare UTM data.

Generation of openUTM

The supply of data from openUTM to SM2 must be generated in the UTM application. The SM2 operand in the MAX statement is provided for this purpose. One of the values ON or OFF must be specified in this operand:

- If MAX...,SM2=ON, delivery of data to openSM2 is activated when the application is started. This can then be deactivated and activated again as necessary during live operation using the UTM administration facilities.
- If MAX...,SM2=OFF is specified, the delivery of data to openSM2 is permitted for this application. It must, however, be explicitly activated during live operation using the UTM administration facilities.

If MAX ...,SM2=NO is generated, openUTM **cannot** deliver any data to openSM2 for this application. It is also not possible to activate the delivery of data using the UTM administration facilities.

Activating the delivery of data to openSM2 using the UTM administration facilities

The UTM administrator can activate the delivery of data to openSM2 using the command KDCAPPL SM2=ON if provision was made for this in the UTM generation (MAX SM2=ON/OFF). KDCAPPL SM2=OFF deactivates the delivery of data.

The UTM administrator can use the KDCINF SYSPARM command to determine whether the application is able to deliver data to openSM2 and whether it is currently delivering data.

It is also possible to activate and deactivate delivery of data to openSM2 using the administration program interface KDCADMI or WinAdmin/WebAdmin.

Requirements in openSM2

Acquisition of the monitoring data is implemented using the INSPECTOR component of openSM2. In order to do this, the UTM applications must be entered in the configuration file of the agent. For information on the precise format of the configuration lines, refer to the online Help system in the INSPECTOR Manager ("The configuration file" section of the relevant agent).



You will find a description of how the monitoring data is output and evaluated in the openSM2 documentation.

12.2 UTM event monitor KDCMON

KDCMON only records UTM events. It is possible to use openSM2 and KDCMON together.

KDCMON can be activated during live operation and then deactivated after the required monitoring period. The data is buffered and written to a file.

The tool KDCEVAL is available for evaluating the data acquired by KDCMON.

12.2.1 Starting and stopping data entry

Data entry can be activated and deactivated using the administration command:

```
KDCDIAG KDCMON={ ON | OFF }
```

This administration function is also available on the KDCADMI program interface and in WinAdmin/WebAdmin.

The UTM administrator can use the following command:

```
KDCINF SYSPARM
```

at any time to determine whether or not data is being recorded.

If openUTM detects that the KDCMON function is not available when it attempts to activate it, then the following message is output to the default destination SYSLOG:

```
K080 KDCMON is not active
```

Possible cause: Communication with the log process has been interrupted (Unix and Linux systems only).

If openUTM detects that the KDCMON function is not available any more while it is acquiring data, then openUTM deactivates the collection of data and informs the user of this fact with message K080.

The following files are generated during collection.

Unix and Linux systems

For each measurement interval openUTM creates a file with the following name:

```
filebase/KDCMON/nnnn.pid
```

where *nnnn* is the sequential number of the measurement interval, starting with 0001 after the application has been started, and where *pid* is the process ID of the utmlog process belonging to the current application run.

Windows systems

openUTM generates a file with the fixed name *filebase*\KDCMON\0001.

This file remains open from the first KDCDIAG KDCMON=ON command until the end of the application run. You can copy this file after you have entered the KDCDIAG KDCMON=OFF command.

Note that the file is empty again after the next KDCDIAG KDCMON=ON command or after the next application start.

12.2.2 Evaluating data with KDCEVAL

The data recorded with KDCMON is evaluated with the KDCEVAL tool. Only the data from *one* application can be evaluated in an evaluation run. KDCEVAL requires several parameters to control the evaluation run. You must enter these parameters after KDCEVAL has been started.

Starting KDCEVAL

You must copy the file that you want to evaluate (*nnnn.pid* in the KDCMON directory, see above) and save it in a file named `eval.in` before you start KDCEVAL.

You then call KDCEVAL as follows:

`utmpath/ex/kdceval` (Unix and Linux systems) or

`utmpath\ex\kdceval` (Windows systems)

After the evaluation program has been started interactively, KDCEVAL outputs the following message to request the input of control parameters:

```
PLEASE ENTER COMMANDS OR 'HELP' OR 'END'
```

KDCEVAL control parameters

The program reads the SYSDTA parameters from *stdin*. The individual commands you can use to control the evaluation have the following format:

APPLINAME applicationname

Name of the application for which the evaluation is to be carried out.

TIME FROM={ t1 | START }, TO={ t2 | STOP }

Time specification for defining the evaluation time limits.

FROM=t1

Start time of the evaluation in seconds.

The time is specified is relative to the time that the data collection was activated (e.g. with the KDCDIAG command).

FROM=START

The evaluation starts at the beginning of the file.

TO=t2

End time of the evaluation.

The time is specified is relative to the time that the data collection was activated (e.g. with the KDCDIAG command).

TO=STOP

The evaluation continues until the end of the file.

The following apply for *t1* and *t2*:

Minimum value: 0

Maximum value: 99999999

LIST { (list₁, list₂,...,list_n [,TABLE]) | (STD [,TABLE]) | (ALL [,TABLE]) }

list₁, list₂,...,list_n

Names of the individual lists to be prepared. The names that you can specify here are indicated on "[Evaluation lists](#)". The TRACE and TRACE2 lists must not be specified at the same time.

STD

This evaluation covers the lists TASKS, SUMM, TIMES and TCLASS.

ALL

The evaluation covers all lists apart from TRACE and TRACE2.

If ALL or STD is specified without TABLE, the round brackets can be omitted.

TABLE

If TABLE is specified in addition, the lists are created in a table format that can be further processed on PC with Excel or another spreadsheet program, see "[Processing evaluation data on the PC](#)". TABLE only works on the segregated lists TASKS, TIMES, TCLASS, TACCL, TACPT and TACLIST.

OPTION **DECIMAL-SEPARATOR**={ COMMA | POINT }, **SHOW-TSN**={ FIRST | ALL }

Defines the decimal separator.

DECIMAL-SEPARATOR=COMMA

The comma is used as the decimal separator.

DECIMAL-SEPARATOR=POINT

The period is used as the decimal separator; this is the default value.

SHOW-TSN=FIRST

In the TRACE2 list, in successive records with identical PID, the PID is only output in the first record in the sequence. The subsequent records in this kind of sequence of records with identical ID contain quotation marks (") in the PID field.

This is the default value.

SHOW-TSN=ALL

In the TRACE2 list, the PID is output in each record. This can be useful if the output list is to be edited or analyzed using a program.

END

This command terminates parameter input.

The HELP command can also be entered with interactive evaluations. The syntax of the commands and the possible list names are output in this case.

Errors and messages

- If one of the commands APPLINAME, TIME or LIST is missing, the evaluation is aborted with the following error message:

```
MANDATORY COMMAND MISSING
```

- In the case of a syntax error, the following message and the incorrect command are displayed:

```
ERROR IN COMMAND
```

- If the time specifications *t1* and *t2* are inconsistent, the following message is output:

```
KDCEVAL: WRONG TIME INPUT
```

- If no records are found in the file for the application or if no data exists within the evaluation time limits, one of the following messages is output:

```
NO EVALUATION : NO RECORD WITH APPLINAME FOUND
```

or

```
NO EVALUATION : NO RECORD IN TIME_INTERVAL
```

- If a DMS error occurs, the following messages are output:

- If KDCEVAL cannot find the `evalin` file:

```
KDCEVAL: NO KDCMON FILE
```

```
KDCEVAL: NO EVALUATION
```

- If the `evalin` file could not be created by KDCMON:

```
NO EVALUATION: NO VALID KDCMON FILE
```

- Version check:

It is only possible to evaluate KDCMON data using KDCEVAL if KDCEVAL has the same openUTM version as the UTM system code. KDCEVAL checks the version of the KDCMON data. If KDCEVAL identifies an illegal version, KDCEVAL aborts the evaluation with the following message:

```
NO EVALUATION: INPUT FILE FROM INVALID UTM VERSION
```

Result of the KDCEVAL evaluation

KDCEVAL writes the evaluation to the file

`kdcmon.appliname`

This file is stored in the current directory.

12.2.3 Processing evaluation data on the PC

If you specify the TABLE operand in addition to the list name in the LIST control parameter for KDCEVAL, the lists are created in table form. This type of processing is only possible for TASKS, TIMES, TCLASS, TACCL, TACPT, and TACLIST lists.

The lists generated in this way can be processed and formatted graphically on the PC using a spreadsheet program such as Excel. The `kdceval.xls` macro is supplied for Excel for this purpose.

Carry out the following steps:

1. Transfer the list file created by KDCEVAL and the `kdceval.xls` macro to a PC.
The macro requires that the file to be evaluated has the suffix `.txt`!
2. Call the `kdceval.xls` macro and read the list file into Excel. Excel then creates a separate spreadsheet for each list, as well as an additional sheet with summary information.
3. Process the individual lists as desired, e.g. by sorting a list and then converting it into a curve chart or bar chart.

12.2.4 Evaluation lists

Each evaluation list includes the following:

- a title containing the name of the evaluation list
- a header, which is identical for all lists
- the specific evaluation list

The list header is structured as follows:

```

NAME OF APPLICATION : appliname      DATE           : day month dd yyyy hh:mm:ss
COMMENCEMENT TIME  : rel-sec SEC.    KDCEVAL VERSION: V07.0A00
END TIME           : rel-sec SEC.    openUTM VERSION: V07.0A00
SYSTEM INFORMATION : system info
APPLICATION INFO   : Appli-Mode S    Cache size,loc  BLKSIZE : 4K    Test-Mode OFF   Data
Compression OFF
    
```

The fields are explained below:

NAME OF APPLICATION	Name of the application.
DATE	Date of data entry with KDCMON.
COMMENCEMENT TIME	Start time of the selected evaluation period (relative to the start time of the data acquisition)
END TIME	End time of the selected evaluation period (relative to the start time of the data acquisition).
SYSTEM INFORMATION	Name and operating system version of the computer and also the execute mode (bit mode 32 bit or 64 bit)
Cache size, loc	Size and storage location of the cache. For storage location, always PS (program space) is output.

In the case of the TRACE and TRACE2 lists, END TIME contains the value 999999 if the entire file is evaluated (parameter TIME FROM=START,TO=STOP).

The processing times are always the ELAPSED TIME (real time).

The following individual evaluations and combinations of evaluations are possible:

TASKS	UTILIZATION OF THE UTM TASKS
SUMM	TRANSACTION EVALUATION
TIMES	DISTRIBUTION OF PROCESSING TIMES
KCOP	KDCS CALLS STATISTIC
WAIT	WAITING TIMES

TCLASS	EVALUATION OF THE TAC CLASSES
TACCL	TAC SPECIFIC TAC CLASS EVALUATION
TACPT	TAC SPECIFIC DISTRIBUTION OF PROCESSING TIMES
TACLIST	TAC SPECIFIC STATISTICS
TRACE	TASK SPECIFIC TRACES
TRACE2	TASK PERFORMANCE TRACES

The individual evaluation lists are described below.

12.2.4.1 TASKS: UTILIZATION OF THE UTM TASKS

This list provides an overview of the utilization levels of the processes of the application. Furthermore, the CPU utilization and the number of input and output operations (I/O's) are indicated for each individual UTM process and the sum is displayed for all tasks of the application.

- 1 = Program
- 2 = System code
- 4 = Bourse Wait

```

|-----|
| PID | START TIME | TASK UTILIZATION , Number Used Tasks: 4 , Number System Tasks: 0
| 9253 | 09:33:05.542 | <1><---2--><-----4----->
| 9250 | 09:32:41.114 | 1><-----2-----><-----4----->
| 9217 | 09:32:41.114 | <-1-><-----2-----><-----4----->
| 9144 | 09:32:39.010 | <1><---2--><-----4----->
|-----|-----|

```

PID	CPU-time	Number I/O	Program	System	Bourse Wait	System Task
9253	19655	16677	13322	32766	323895	N
9250	16683	14145	11206	54501	328704	N
9217	33065	27990	22564	60260	311587	N
9144	19863	16575	13607	41018	341891	N
Summ	89266	75387				

Explanation of the terms in the list:

PID	Process ID of the UTM process.
START TIME	Time of the first record of this process (absolute).
Program	Proportion of processing time of the application program in the UTM process.
System code	of processing time of the UTM system code.
Bourse Wait	Proportion of time awaited by the process for new jobs to enter the job queue.
System Task	Specifies whether this process is a UTM system process.

The times output in the columns Program, System, Database and Bourse Wait are real times. The unit used is milliseconds (in the same way as for the CPU time).

A reduction in the number of tasks during the evaluation time limits must be avoided for the TASKS evaluation as this would lead to distorted results. In this case you should use other evaluation time limits.

12.2.4.2 SUMM: TRANSACTION EVALUATION

This list provides an overview of the services and transactions for the evaluation period. The list only includes transactions that lie completely within the evaluation period. The evaluation tool KDCEVAL also indicates the CPU utilization of all program unit runs that were terminated within the evaluation time limits:

COUNT OF TRANSACTIONS	:	19126
COUNT OF SERVICES	:	3059
COUNT OF DIALOG STEPS	:	19126
NUMBER OF DIALOG STEPS PER SECOND	:	59,91
TOTAL CPU-TIME USED IN MSEC	:	89094 1)

- 1) This line indicates the total CPU utilization of the program unit runs. This also includes the utilization in the UTM and operating system code, insofar as this occurs within the program unit runs, as well as the start and end processing of program unit runs in openUTM. Other actions of the UTM tasks that do not belong directly to program units are not included.

12.2.4.3 TIMES: DISTRIBUTION OF PROCESSING TIMES

In tabular form, this list indicates a distribution of processing times for the program units. These times do not include the wait time before processing by openUTM.

The list has the following format:

PROCESSING TIMES (MSEC)	NUMBER	PERCENT
0 - 100	21721	99,62
101 - 200	2	0,00
201 - 500	0	0,00
501 - 1000	0	0,00
1001 - 2000	80	0,36
2001 - 5000	0	0,00
5001 - 10000	0	0,00
10001 - 20000	0	0,00
20001 - 50000	0	0,00
50001 - 100000	0	0,00
> 100000	0	0,00

This list indicates the number of complete program unit runs and the percentage for the respective time class.

12.2.4.4 KCOP: UTM CALLS STATISTIC

This table specifies how often the UTM calls occurred in the evaluation period.

Calls that are not included in the list of calls known to KDCEVAL appear under *others*.

This list contains calls that are issued by openUTM for internal processing and are not available to the user:

CONT	Call following formatting or internal database communication.
ADMI	UTM administration action
WAIT	End of processing of a program run.
NOOP	The buffer area of MESSAREA must bse flushed.

The KCOP list has the following format

OP	OM	NUMBER	OP	OM	NUMBER
ADMI		7	MPUT	HM	0
APRO	AM	0	MPUT	ID	0
APRO	DM	0	MPUT	NE	6
APRO	IN	0	MPUT	NT	24378
CONT		19019	MPUT	PM	0
CTRL	AB	0	MPUT	RM	0
CTRL	EC	0	NOOP		0
CTRL	PE	0	PADM	AC	0
CTRL	PR	0	PADM	AI	0
CTRL	SC	0	PADM	AT	0
DADM	CS	0	PADM	CA	0
DADM	DA	0	PADM	CS	0
DADM	DL	0	PADM	OK	0
DADM	MA	0	PADM	PI	0
DADM	MV	0	PADM	PR	0
DADM	RQ	0	PEND	ER	0
DADM	UI	0	PEND	FC	0
DGET	BF	0	PEND	FI	3060
DGET	BN	0	PEND	FR	0
DGET	FT	0	PEND	KP	0
DGET	NT	0	PEND	PA	2677
DGET	PF	0	PEND	PR	0
DGET	PN	0	PEND	PS	0
DPUT	NE	0	PEND	RE	16067
DPUT	NI	0	PEND	RS	0
DPUT	NT	0	PEND	SP	0
DPUT	RP	0	PGWT	CM	0
DPUT	QE	0	PGWT	KP	0
DPUT	QI	0	PGWT	PR	0
DPUT	QT	0	PGWT	RB	0
DPUT	+I	0	PGWT	RT	0
DPUT	-I	0	PGWT	ST	0
DPUT	+T	0	PTDA		144
DPUT	-T	0	QCRE	NN	0
FGET		145	QCRE	WN	0
FPUT	NE	225	QREL	RL	0
FPUT	NT	2	RSET		4

FPUT RP	0	SGET GB	2678
FPUT UF	0	SGET KP	0
GTDA	1	SGET RL	0
INFO CD	0	SGET US	0
INFO CK	0	SIGN CK	0
INFO DT	0	SIGN CL	0
INFO FH	0	SIGN CP	0
INFO GN	0	SIGN OB	0
INFO LO	0	SIGN OF	0
INFO PC	0	SIGN ON	0
INFO SI	0	SIGN ST	0
INIT	21804	SMSG	0
INIT PU	0	SPUT DL	0
INIT MD	0	SPUT ES	0
LPUT	0	SPUT GB	2677
MCOM BC	0	SPUT MS	0
MCOM EC	0	SPUT US	0
MGET	18981	SREL GB	0
MGET NT	0	SREL LB	0
MPUT CM	0	UNLK DA	0
MPUT EM	0	UNLK GB	0
MPUT ES	0	UNLK US	0
MPUT GC	0	WAIT	19153
OTHERS	0		

12.2.4.5 WAIT: WAITING TIMES

To establish bottleneck situations, openUTM inserts measuring jobs into the job queue at regular intervals if KDCMON is activated. The wait time of the jobs in the UTM queue can be determined on the basis of the time at which the job was introduced (absolute time stamp) and the time of processing. The time difference between the individual pseudo jobs is approximately 10 seconds.

The following information is logged in the WAIT list:

- The WAITING TIME column indicates the established wait time for each pseudo job in seconds.
- For these wait times, the evaluation tool KDCEVAL also calculates the maximum, minimum, and mean value in seconds and outputs these values under UTM WAITING TIMES.
- The NUMBER OF TASKS column indicates the number of processes available in the application at this time. The UTM system processes are not included in this number.

If the wait time is too long, the number of UTM tasks should be increased.

The WAIT list has the following format:

```
+-----+
|  TIME STAMP      |  WAITING TIME |  NUMBER OF TASKS |
+-----+
|  09:32:41.114   |      0,000   |          4        |
+-----+
|  09:32:51.114   |      0,000   |          4        |
+-----+
|  09:33:01.114   |      0,018   |          4        |
+-----+
|  09:33:11.534   |      0,000   |          4        |
+-----+
|  09:33:21.534   |      0,008   |          4        |
+-----+
|  09:33:31.534   |      0,000   |          4        |
+-----+
|  09:33:41.534   |      0,000   |          4        |
+-----+

                UTM WAITING TIMES:
TIME STAMP : 09:33:01.114   WAITING TIME MAXIMUM :      0,018
TIME STAMP : 09:33:41.534   WAITING TIME MINIMUM :      0,000
NUMBER OF ENTRIES:      7   WAITING TIME AVERAGE :      0,004
```

12.2.4.6 TCLASS: EVALUATION OF THE TAC CLASSES

The TCLASS list contains an overview of job processing of TACs in the individual TAC classes (1 through 6) in tabular form. In the evaluation, all dialog TACs to which no TAC class was assigned during generation with KDCDEF are combined into TAC class 0.

In the UTM generation, the user can define the maximum number of tasks that can operate for a TAC class at any one time. When this number is reached, subsequent jobs are placed in a TAC class-specific queue.

TAC- CLASS	NUMBER CALLS	DISTRIBUTION IN PERCENT			AVERAGE WAIT TIME (IN MSEC)	MAXIMUM WAIT TIME (IN MSEC)	MINIMUM WAIT TIME (IN MSEC)
		NUMBER CALLS	WAIT- TIME=0	WAIT- TIME>0			
0	10	0,04					
1	21646	99,27	97,90	2,10	184	1010	1
2	0	0,00			0	0	0
3	3	0,01	66,66	33,34	296	296	296
4	0	0,00			0	0	0
5	0	0,00			0	0	0
6	0	0,00			0	0	0
7	0	0,00			0	0	0
8	0	0,00			0	0	0
9	145	0,66	2,75	97,25	1	2	1
10	0	0,00			0	0	0
11	0	0,00			0	0	0
12	0	0,00			0	0	0
13	0	0,00			0	0	0
14	0	0,00			0	0	0
15	0	0,00			0	0	0
16	0	0,00			0	0	0
			97,26	2,74	141		

21659 DIALOG TACS WERE CALLED
145 ASYNCHRONOUS TACS WERE CALLED

The TCLASS list contains the following information:

- The NUMBER CALLS column indicates the number of TAC calls in the evaluation period for a TAC class.
- The DISTRIBUTION IN PERCENT column contains percentage values.
The subcolumn NUMBER CALLS specifies the percentage of calls of a TAC class within the number of all TAC calls. The next two columns contain a percentage breakdown of the calls of this TAC class into the following categories:
 - calls that were processed immediately (WAITTIME=0), and
 - calls that had to be placed in a TAC class-specific queue (WAITTIME>0).
- The values in the columns AVERAGE / MINIMUM / MAXIMUM WAIT TIME refer to the jobs which openUTM temporarily placed in a TAC class-specific queue. The average minimum or maximum wait time of a job per TAC class is displayed.

i The average wait time of jobs per TAC class can also be queried with the administration command `KDCINF TACCLASS` or with the corresponding function in WinAdmin/WebAdmin or KDCADMI while an application is running.

Wait time for dialog jobs

In the case of dialog jobs, the wait time is the period between the acceptance of the job by the application (job retrieved from the queue of the application) and the start of the program unit. Displacement can also occur between individual program units.

Wait times for asynchronous jobs

openUTM also records the wait time of asynchronous jobs. The wait time is defined as follows:

Asynchronous job	Definition of “wait time”
Input asynchronous TAC	Period between the acceptance of the job by openUTM and the start of the asynchronous service.
FPUT call in the program unit	Period between the end of the transaction in which the FPUT job was executed, and the start of the asynchronous service.
DPUT job in the program unit	Period between the conversion of the DPUT into FPUT and the start of the asynchronous service.

If the asynchronous job was not created in the current application run, the asynchronous wait time is always taken to be the time difference between the start of the application and the start of the asynchronous job.

12.2.4.7 TACCL: TAC SPECIFIC TAC CLASS EVALUATION

The TACCL list contains the same information as the TCLASS list, except that it is broken down according to the individual transaction codes. It lists all TACs that were called in the evaluation period. The TACs are listed in the sequence they first occurred. For an explanation of the individual columns, see the description of the TCLASS list format.

TAC	TAC-CLASS	NUMBER CALLS	DISTRIBUTION IN PERCENT			AVERAGE WAIT TIME (MSEC)
			NUMBER CALLS	WAIT-TIME=0	WAIT-TIME>0	
CVARL	1	2678	12,28	98,91	1,09	356
CVARL1	1	16066	73,68	98,91	1,09	355
CVAR1	1	2677	12,27	99,02	0,98	358
KDCINF	3	2	0,00	100,00	0,00	0
UPDEMP	0	3	0,01			
PTDA	1	144	0,66	0,69	99,31	11
PTDAA	9	144	0,66	2,77	97,23	1
...
...

No WAIT TIME specifications are entered for TACs of TAC class 0.

12.2.4.8 TACPT: TAC SPECIFIC DISTRIBUTION OF PROCESSING TIMES

This table lists the minimum (MIN), maximum (MAX), and mean (MEAN) processing time for all TACs processed within the evaluation period. It only includes the TACs whose start and end time lie within the evaluation period. The list has the following format:

TAC	PROCESSING TIME PER TAC (IN MSEC)		
	MEAN	MIN	MAX
UPDEMP	28	13	58
CVAR1	14	0	50
KDCINF	11	11	12
PTDA	8	7	24
CVARL	6	6	114
PTDAA	5	5	12
CVARL1	5	0	151
...	.	.	.
...	.	.	.

The table is sorted in descending order according to the mean processing times. Only TACS with a mean processing time > 0 are displayed.

12.2.4.9 TACLIST: TAC SPECIFIC STATISTICS

This list contains the following TAC-specific information:

- The average size of the communication area (column AVERAGE SIZE OF KB)
- The breakdown of processing time into:
 - 1: program
 - 2: system code

The list has the following format:

```
-----  
| TAC | NUMBER CALLS | AVERAGE SIZE OF KB |  
-----  
| CVARL | 2678 | 956 | <-----1-----><-----2----->  
-----  
| CVARL1 | 16066 | 956 | <-----1-----><-----2----->  
-----  
| CVAR1 | 2677 | 956 | <---1---><-----2----->  
-----  
| KDCINF | 2 | 0 | <-----1-----><-----2----->  
-----  
| UPDEMP | 3 | 0 | <-----1-----><---2--->  
-----  
| PTDA | 144 | 1 | <-----1-----><-----2----->  
-----  
| PTDAA | 144 | 1 | <-----1-----><-----2----->  
-----  
| ... | . | . |  
-----  
| ... | . | . |  
-----
```

The list is not sorted; the TACs appear in the sequence in which they first occur in the file.

The list only includes TACs whose start and end times lie within the analysis period.

12.2.4.10 TRACE: TASK SPECIFIC TRACES

TRACE lists can be created for a more precise analysis of the execution of a UTM application. This list contains all UTM calls for the individual UTM processes in chronological order.

The TRACE list only ever contains the data for the first 6 processes. If data from more than 6 processes exists for the evaluation period, the TRACE2 table should be used for evaluation.

The list is sorted in chronological order.

The TIME STAMP column contains the time stamp of the corresponding call that was logged (to the nearest millisecond).

The TRACE list records the following events and data:

- The transaction code called (TAC).
- The transaction ID. In openUTM, a unique transaction ID is assigned to each transaction. This identifier is also transferred to the attached databases on the UTM-DB interface. In this way, it is possible to link database traces with these UTM traces and establish relationships between UTM and DB processes. The transaction ID is made up of four parts:

SC	Session counter: This numbers the application runs. The number is 1 after a regeneration, and is incremented by 1 each time the application starts.
VC	Service counter: This numbers the services within the application run and runs up to 16 777 216 (2^{24}).
TC	Transaction counter: This numbers the transactions within a service and runs up to 32 768 (2^{15}).
VN	Conversation number: This is the number of an internal UTM table for service administration.

These four parts are logged after the KDCS call INIT.

The VC and TC specifications are of interest to the user.

- All UTM calls with operation modifications. Internal UTM calls (WAIT, CONT, ...) are also listed. See the KCOP list.

The following are also logged:

- KCMF for KCMF-relevant calls
- KCRN for KCRN-relevant calls
- KCLT for PADM/DADM calls
- In the event of an abort with PEND ER/ FR as diagnostic information:
 - the TAC of the program unit that caused the abort
 - the return codes KCRCDC and KCRGCC
 - VC and TC for the assignment to the aborted service
- With a PEND RS as diagnostic information:
 - the TAC of the current program unit
 - VC and TC for the assignment to the aborted service

As long as no process switch takes place, all calls for processing a dialog step are listed in succession in the same PID column. Following a PEND PA/PR/SP, a process switch can only occur with a change of TAC class. The interruption of a process by the operating system can be seen by the fact that the calls are continued in another process column midway through the processing of a dialog step.

Example

TIME STAMP	PID :9144	PID :9217	PID :9250	PID :
9253	PID :	...		
09:39:11.454+-		--+	--+	-
CVARL1 +-	...			
09:39:11.454				
INIT		...		
09:39:11.454				SC :
1	...			VC :
09:39:11.454				TC :
97829	...			VN :
09:39:11.454				-
3	...	--+	--+	MGET
09:39:11.454+-	1052 +- ...			
09:39:11.454				
@@		...		
09:39:11.454	CONT			
		...		
09:39:11.454				MPUT
NT		...		
09:39:11.457+-		--+	--+	- PEND RE
CVARL1 +-	...			
09:39:11.462				
WAIT		...		
09:39:11.463	GTDA			
		...		
09:39:11.463	INIT			
		...		
09:39:11.463	SC :	1 +-	--+	--+
--+	...			
09:39:11.463	VC :	97925		
		...		
09:39:11.463	TC :	1		
		...		
09:39:11.463	VN :	149		
		...		
09:39:11.463			CONT	
		...		
09:39:11.463	MGET @@	+-	--+	--+
--+	...			
09:39:11.463	FPUT NE GTDAA			
		...		
09:39:11.463	MPUT NT			
		...		
09:39:11.465	PEND FI			
		...		
09:39:12.471	WAIT	+-	--+	--+
--+	...			
09:39:12.472				CVARL1
		...		
09:39:12.472			INIT	
		...		
09:39:12.472			SC :	1
		...		
09:39:12.472+-		--+	-	VC : 97829 +-
--+	...			

```

09:39:12.472|          |          | TC :          4
|          |          |          |
09:39:12.472|          |          | VN :          1052
|          |          |          |
09:39:12.472|          |          | MGET @@
|          |          |          |
09:39:12.472| CONT      |          |
|          |          |          |
09:39:12.472+- -+-      |          | MPUT NT          +-
-+-      ...
09:39:12.474|          |          | PEND RE CVARL1
|          |          |          |
09:39:12.479|          |          | WAIT
|          |          |          |
09:39:12.480|          | GTDA    +-      -+-      -+-
-+-      ...
09:39:12.480| INIT      |          |
|          |          |          |
09:39:12.480| SC :      |          1 |          |
|          |          |          |
09:39:12.480| VC :      |          97957 |          |
|          |          |          |
09:39:12.480| TC :      |          1 |          |
|          |          |          |
09:39:12.480| VN :      |          149 +-      -+-      -+-
-+-      ...
09:39:12.480|          |          |          |
CONT      |          |          |
09:39:12.480| MGET @@   |          |
|          |          |          |
09:39:12.480| FPUT NE GTDAA |          |
|          |          |          |
09:39:12.480| MPUT NT   |          |
|          |          |          |
09:39:12.480|          |          |          |
|          |          |          |
09:39:12.482| PEND FI   |          +-      -+-      -+-
-+-      ...
09:39:13.488| WAIT      |          |
|          |          |          |

```

12.2.4.11 TRACE2: TASK PERFORMANCE TRACE

The most important events in the program units of the applications are contained in the TRACE2 evaluation list in sequence. Since the evaluation is not broken down into columns for the UTM tasks as in the TRACE list, the TRACE2 list can show any number of tasks. In addition to the entries of the TRACE evaluation, TRACE2 also contains important data for performance analysis.

The entries in the evaluation are sorted in chronological order. The TIME STAMP column contains the time stamp of the event (accuracy: milliseconds).

The TRACE2 evaluation list records the following events and data:

- Start of a program unit as an entry `strt >>> tac` with
 - Transaction code of the program unit
 - TAC class
 - Current I/O and CPU stamp
 - Wait time of job in the TAC class
- All UTM function calls with operation code and modification, plus the information:
 - KCMF for KCMF-relevant calls
 - KCRN for KCRN-relevant calls
 - KCLT for PADM and DADM calls
 - For PEND calls with KCOM = ER/FR/RS, the transaction ID (SC,VC,TC, and VN) for the assignment to the aborted service.
 - If KCRCCC != 0, the return codes KCRCDC and KCRGCC and the transaction ID (SC,VC,TC and VN).
- End of the program unit as an entry `WAIT end<<<<` with
 - CPU utilization in the program unit in microseconds in the "CPU" column
 - I/O utilization in the program unit in column "I/O"
 - At the end of column "TRACE" the first two bytes of the last bourse announcement are rendered as four characters representing the sedecimal value of the announcement.

This information allows the system service a better analysis and diagnosis of possible performance problems.

The structure elements <<<<<< in the list make it easier to read the entries.

Example

TIME STAMP	TRACE	PID	SC	VC	TC	VN KCRN	CPU	KCMF	I/O	TACCLASS	Q.TIME
16:39:38.339+	INIT	-----+	"	-----+							
16:39:38.339	MGET	-----	"								
16:39:38.339	ADMI	-----	"								
16:39:38.339	ADMI	-----	"								
16:39:38.339+	MPUT NT	-----+	"	-----+							
16:39:38.339	MPUT NT	-----	"								
16:39:38.339	MPUT NT	-----	"								
16:39:38.339	MPUT NT	-----	"								
16:39:38.339+	MPUT NT	-----+	"	-----+							
16:39:38.339	MPUT NT	-----	"								
16:39:38.339	MPUT NT	-----	"								

13 Load simulation with Workload Capture and Replay

Thanks to the Workload Capture & Replay function, it is possible to record UTM application communications with UPIC clients and then replay these in combination with adjustable load profiles. In this way, it is possible to test the behavior of the UTM application at high loads under real-life conditions.

Workload Capture & Replay consists of the following components:

- *UPIC Capture*: Records communication with the UPIC client.

The trace function BTRACE (BCAM trace), which is present on all the server platforms, is used to record UPIC sessions.

It may then also be necessary to merge the traces.

- *UPIC Analyzer*: Used to analyze the recorded communication.

Analysis is performed using the program *UPICAnalyzer* which is supplied with UPIC on 64-bit Linux systems.

- *UPIC Replay*: Used to replay the recorded UPIC session with different load parameters (speed, number of clients).

This is done using the program *UPICReplay* which is supplied with UPIC on 64-bit Linux systems.

i The *UpicAnalyzer* program must be compatible with the openUTM version which has been used for capturing. „openUTM-Client V7.0 for the UPIC carrier system“ is compatible with openUTM V7.0, for example.

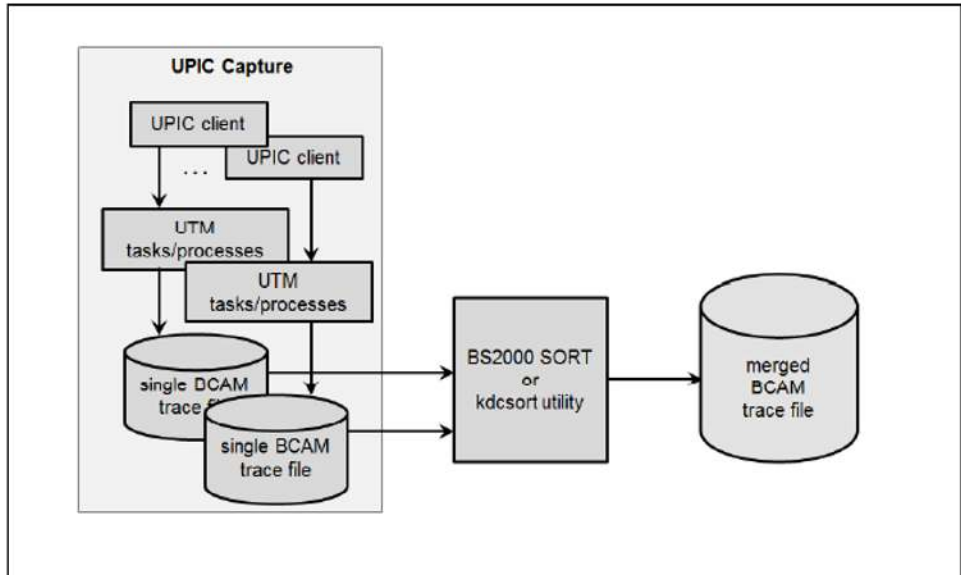
The version of the *UpicReplay* program can only process input files which have been created using the same version of the *UpicAnalyzer* program.

In addition, the utility program *kdcsort* is supplied on Unix, Linux and Windows systems in order to sort the recordings of communications performed in multiprocess operation.

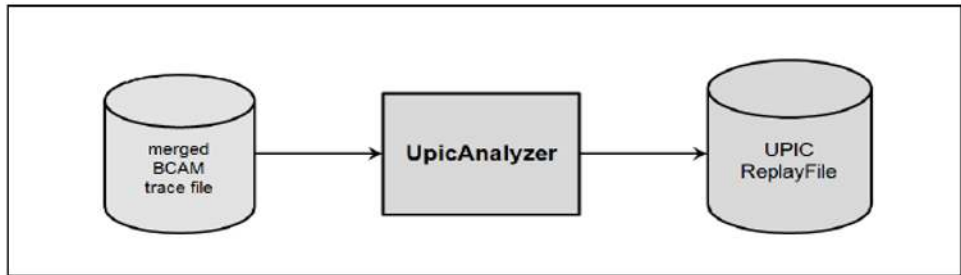
You perform the following steps to run the Workload Capture & Replay function:

1. Enable the BCAM trace and start UPIC communication, see [section “Recording the UPIC conversation \(UPIC Capture\)”](#).
2. Stop the BCAM trace and merge the BCAM trace entries in a trace file (if necessary), see [section “Merging trace entries”](#).

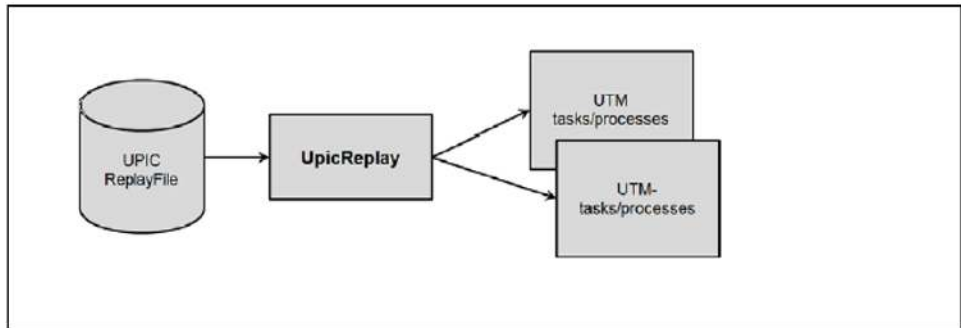
These two steps are illustrated in the figure below.



3. Perform a binary transfer of the trace file to the 64-bit Linux system where you have installed UPIC. Please note that you have to transfer the file by means of openFT if the file is transferred from a BS2000 system.
4. Create a UPIC ReplayFile on the 64-bit Linux system on which the UPIC client is installed. To do this, call the program *UpicAnalyzer* with the trace file as input file, see the figure. For details, see [section "Preparing data using the program UpicAnalyzer"](#).



5. Start the program *UpicReplay* with the UPIC ReplayFile as the input file, see the figure. For details, see [section "Replaying the UPIC session using the program UpicReplay"](#).



13.1 Recording the UPIC conversation (UPIC Capture)

For this step, the UTM application can be run on any UTM platform. (BS2000, Unix, Linux, or Windows system).

The UPIC clients can run on any UPIC platform. Even UPIC clients based on the product openUTM-JConnect are supported.

During this phase, the communication between the UTM application and the UPIC clients must be recorded in full and the trace length must be greater than the maximum message length. This is achieved using the UTM function BCAM trace.

i If both the UPIC clients and the UTM application are supporting encryption then UPIC conversations for users with password cannot be replayed with *UpicReplay*. The same applies to UPIC conversations with user data encryption.

In addition, *UpicReplay* ignores the UPIC client call for setting a new password.

On BS2000 systems, both UPIC clients and UTM applications support encryption.

On Unix, Linux and Windows systems, UPIC clients and UTM applications only support encryption if a suitable openssl library has been provided.

Please note that it must also be possible to repeat the required UTM services as often as necessary.

To do this, proceed as follows:

1. Start the BCAM trace by setting the start parameter `BTRACE=ON,length`, see "[Start parameters for openUTM](#)". You are recommended to specify the maximum value for *length* to prevent messages from being truncated. You can also enable the BCAM trace by means of the administration functions (KDCDIAG command or via WinAdmin/WebAdmin). In this case, however, the default value (256 bytes) is assumed for *length*, if you have not specified any other length with the BTRACE start parameter.
2. Perform the UPIC conversations between the UPIC client and the UTM application that are required for the load simulation. This also includes all aspects of establishing the connection to the UPIC clients. The associated UTM services must be fully completed at least once.
3. End the BCAM trace by means of the KDCDIAG command or via WinAdmin/WebAdmin.

This step results in binary TRACE files for all UTM processes. For details on the BTRACE files, see openUTM manual "Messages, Debugging and Diagnostics".

13.2 Merging trace entries

This step is necessary if the UTM application was running with more than one process during recording, a scenario that generally applies in the case of UTM applications running at medium or high load.

In this step, the binary BTRACE files of all UTM processes are sorted and entered in a common BTRACE file on the basis of their timestamps. This process step must always run on the same platform as step 1 (UPIC Capture).

On Unix, Linux and Windows systems, you must use the UTM utility program *kdcsort* to perform this step, see below.

This step results in a sorted binary BTRACE file that contains all the trace entries in the correct temporal sequence.

You can use the provided BTRACE example procedure for sorting.

Utility program *kdcsort*

The utility program *kdcsort* reads the trace entries from multiple BTRACE files and writes the trace records in the correct temporal sequence to an output file. It is started as follows:

On Unix and Linux systems, from the shell with

```
utmpath/ex/kdcsort btrace_out btrace-1 btrace-2 ... btrace-n
```

On Windows systems, from a command prompt window with

```
utmpath\ex\kdcsort btrace_out btrace-1 btrace-2 ... btrace-n
```

Meaning of the parameters

btrace_out Name of the output file to which the sorted trace records are to be written.

btrace-1 *btrace-2* ... *btrace-n*

Names of the recorded BTRACE files. At least two files must be specified.

The file names must be specified separated by spaces.

The output file from *kdcsort* can then either be prepared as a common list for all UTM work processes using the UTM utility program *kdcbrtc* or be further processed using the program *UpicAnalyzer*.

13.3 Preparing data using the program *UpicAnalyzer*

The program *UpicAnalyzer* is supplied with UPIC on Linux (64-bit). *UpicAnalyzer* reads the trace records from a BTRACE trace, filters out the UPIC trace records, prepares these and writes them to a file in a specific format (UPIC ReplayFile Layout). This file can then be used as the input file for the program *UpicReplay*.

UpicAnalyzer is called as follows from the Linux shell:

```
UpicAnalyzer inputfile outputfile
```

Meaning of the parameters

inputfile

Name of the BTRACE file that you have transferred to the Linux system.

outputfile

Name of the output file (UPIC ReplayFile). You can use this file to play back the UPIC session with *UpicReplay*.

The program *UpicAnalyzer* recognizes the type of platform on which the trace file was created and processes the contents in the light of the platform's specific characteristics.

Example

The transferred trace file has the name *btrc.sorted*. It has to be prepared and the output written to the file *Replayfile*. The call is as follows:

```
UpicAnalyzer btrc.sorted Replayfile
```

Output:

```
Program "UpicAnalyzer": Version 7.0 build yyyy-mm-dd on Linux Intel    ,64 Bit ,  
Little-Endian started  
  
inputfile "btrc.sorted"  
  
outputfile "Replayfile"  
  
109 UTM BCAM trace records with 17218 bytes read.  
  
25 UPIC replay records with 2046 bytes written.  
  
Program "UpicAnalyzer" finished.
```

13.4 Replaying the UPIC session using the program `UpicReplay`

The program `UpicReplay` is a UPIC client program that is supplied with UPIC on Linux (64-bit). Before replaying the session, you may need to adapt the UPIC configuration and/or the generation of the UTM application.

To replay the session, you should use the same UTM platform as for recording. Exceptions are possible, see [“Different platforms for Capture and Replay” \(Adapting the UPIC configuration and UTM generation\)](#).

13.4.1 Adapting the UPIC configuration and UTM generation

To perform the operation on a Linux system, you need the side information file *upicfile* containing at least one entry with the name UPREPLAY. The entry must have the prefix SD or ND. For exceptions, see [“Different platforms for Capture and Replay” \(Adapting the UPIC configuration and UTM generation\)](#).

This entry must be a valid entry with the TAC of a service of the UTM application. (e.g. "DEMO"). The program *UpicReplay* uses this entry to address the UTM application. The program *UpicReplay* may set the TAC appropriately using data from the replay file.

Example of a upicfile entry

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

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

UTMTEST1 must have been generated either in MAX APPLNAME or in a BCAMAPPL statement.

Notes on UTM generation

During the UPIC Replay step, and in particular in the case of high load, the UTM application may need to permit more UPIC connections from the program *UpicReplay* than were originally present during recording. Consequently, it is advisable to use an adequately dimensioned UPIC terminal pool with multiconnect functionality for UPIC access, e.g.:

```
TPOOL LTERM=REPL, PTYPE=UPIC-R, CONNECT=MULTI, NUMBER=1000
```

In this case, up to 1000 UPIC clients can sign on simultaneously via the terminal pool.

If the UPIC Replay step runs at high load then it may be necessary to increase load-dependent generation parameters. In particular, you must pay attention to the following:

- The UTM cache must be sufficiently large (MAX CACHESIZE)
- The page pool must be sufficiently large (MAX PGPOOL)
- The number of UTM processes must be sufficient (MAX TASKS)
- The number of permitted concurrent users must be sufficiently large (MAX CONN-USERS)

Different platforms for Capture and Replay

During replay, the data is transferred 1:1 to the UTM application. If the data includes, for example, hardware-dependent binary data, then this leads to errors if there is a change of platform. Consequently, the following applies:

- It is not possible to record a UTM application session on BS2000 and then replay this with a UTM application on a Unix, Linux or Windows system. Reason: The data in the trace file is present in EBCDIC format and conversion to ASCII is not supported in UPIC.
- It is not possible to switch between 32-bit and 64-bit platforms even within one and the same family of platforms.
- It is possible to record a UTM application session on a Unix, Linux or Windows system and then subsequently play this back using a UTM application on a BS2000 system. One prerequisite is that only pure ASCII text data is transferred during the session.

In this case, you must enter HD as the prefix in the *upicfile* in order to ensure that the data is converted correctly between ASCII and EBCDIC.

13.4.2 Calling UpicReplay

UpicReplay plays the recorded UPIC conversations back again, see “[Functioning of UpicReplay](#)”. During this step, log messages and warnings are output to *stdout* and debugging or error messages are output to *stderr*.

UpicReplay is called as follows from a Linux shell:

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

Meaning of the parameters

InputFileName

Name of the UPIC ReplayFile that you have created with UpicAnalyzer.
Mandatory parameter.

-c<numberOfClients>

numberOfClients specifies the number of UPIC clients for which the recorded conversations are to be replayed.

Default: 1, (corresponds to -c1) i.e. only one client is simulated.

The actual limit depends on the relevant system limit.

-s<speedPercentage>

speedPercentage specifies the replay speed as a percentage of the original speed. This makes it possible to simulate long and short thinking times.

Default: 100 (corresponds to -s100) i.e. original speed.

-s200 means 200%, i.e. twice the speed, achieved by halving the thinking time.

-d Enable debug output to *stderr*, i.e. debug messages are output on thread generation and there are fewer messages on send and receive calls.

-dd Enables extended debug output to *stderr*, i.e. detailed debug messages are output. This option is only intended for internal *UpicReplay* diagnoses.

-dd is only of value when simulating a small number of clients.

Default: no debug output.

Example

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

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

13.4.3 Functioning of UpicReplay

Whenever possible, *UpicReplay* replays the communication exactly as it was during recording:

- A UPIC thread that replays the relevant UPIC conversation of the UPIC client is generated for each UPIC PTERM /LTERM for which a trace record is found in the UPIC ReplayFile.
- This UPIC thread runs in a loop that sends all the input messages to the UTM service in the same way as during recording, i.e. with the same data content and control flow. The procedure is similar for the retrieval of output messages from the UTM application. In this case, the content of the output messages is not checked.

Problems on replay

In the following cases, discrepancies occur between the recording and its reproduction on replay:

- Incomplete recording

A UPIC conversation (i.e. a UTM service) was started before recording via BCAM trace was activated.

A corresponding message is output and all input messages from this started conversation for this client are discarded.

The UPIC client then searches the recording for the start of a new conversation for this UPIC client:

- If a new conversation is found in the records recorded for this PTERM/LTERM then this client first waits in accordance with the recorded timestamps and then starts the load simulation from this point.
- If no new conversation is found then this UPIC replay thread is terminated without communication with the UTM application.

- Truncated service

During replay, a UTM service is terminated (normally or abnormally) after fewer communication steps than in the recording of the UTM application. This can occur:

- if the application program is not able to process the recorded input data correctly because, for example, the input message contains time specifications which are rejected by the program as "late". The UTM service is therefore terminated prematurely.
- if an impermissible UTM transfer admission is used during replay, e.g. missing UTM administration authorization.

In this case, a corresponding message is output and the UPIC Replay thread rejects further messages until it finds a new start of conversation for this client in the recording. The UPIC thread then continues at this start of conversation following a corresponding pause time or it terminates if no further conversation is found for this UPIC client.

- Conversation too long

On replay, a UTM service has more communication steps than during recording.

The UPIC thread terminates this service abnormally by disconnecting the connection due to unrecorded input data. It also generates a specific warning.

The start of the next conversation for this client is then searched for in the recording. The UPIC thread then continues at this start of conversation following a corresponding pause time or it terminates if no further conversation is found for this UPIC client.

- Incomplete input message

An input message could not be fully recorded due to the trace record length restriction, despite an effort to compress it during recording.

The record is rejected with a warning and the start of the next conversation for this client is searched for in the recording.

The UPIC thread then continues at this start of conversation following a corresponding pause time or it terminates if no further conversation is found for this UPIC client.

- Other errors

Another, unexpected return code, not covered by the cases listed above, is reported at the UPIC program interface.

This situation can occur, for example, if the UTM application is either inaccessible or rejects the establishment of a connection.

In such cases, the UPIC thread outputs an error message.

The relevant UPIC thread is terminated without searching for new conversations for this client in the recording. All other UPIC conversations that are not directly affected by this problem continue to run unchanged.

14 Appendix

- Installing openUTM on Unix and Linux systems
 - Installing UTM system functions on Unix and Linux systems
 - Using different socket network processes (Unix and Linux systems)
 - Installing an openSM2 connection (Unix and Linux systems)
- Installing openUTM on Windows systems
 - Installation of openUTM-Server (Windows systems)
 - User environment (Windows systems)
- Structure of the openUTM installation directory
- Environment variables of a UTM application
 - General environment variables for openUTM
 - Environment variables for work processes
 - Environment variables for using openSSL
 - Environment variables for the KDCDUMP tool
 - Environment variables for the X/Open interface XATMI
 - Additional environment variables for openUTM on Unix and Linux systems
 - Additional environment variables for openUTM on Windows systems
- Structure of the accounting records of openUTM
 - Structure of an accounting record
 - Structure of a calculation record
- Processing print output without printer control (Unix and Linux systems)
- Sample programs and sample applications
 - Sample programs for a publish / subscribe server
 - Sample program for moving messages from the dead letter queue selectively
 - Sample programs for HTTP clients
 - CPI-C sample programs
 - Sample procedures on Unix and Linux systems
 - Sample procedures on Windows systems
 - Sample application on Unix and Linux systems
 - openUTM Quick Start Kit on Windows systems

14.1 Installing openUTM on Unix and Linux systems

Before you can create and run UTM applications on your system, openUTM itself must be installed in the system.

“openUTM” is understood to be the UTM system functions (system code), C includes, and COBOL COPYs for creating the UTM main routine, programs for the dialog terminal processes, the print processes, the timer process (timeout), the network processes and the tools for creating, operating and modifying UTM applications.

i If you are using a number of hard disks on your processor, the UTM application and the database system should be contained on different disks for performance reasons.

The C++ connection module is also compiled at installation.

14.1.1 Installing UTM system functions on Unix and Linux systems

The operating system determines how you must install openUTM on your system. See the Release Notice and delivery information for further information on the installation. The Release Notice also lists the version dependencies on products that work with openUTM. The installation commands are listed in the delivery information.

utmpath

In this manual the directory containing the files needed to run openUTM is referred to as the **utmpath**.

The following applies for the path name of *utmpath*.

- If openUTM is shipped on a platform in 32-bit mode, the following directory is created at installation.
utm-installationdirectory/utm70a00/32 for 32-bit mode
- If openUTM is shipped on a platform in 64-bit mode, the following directory is created at installation.
utm-installationdirectory/utm70a00/64 for 64-bit mode
- If openUTM is shipped on a platform in 32-bit and 64-bit mode, both directory trees are created at installation.

utm-installationdirectory is the directory specified at installation. *utm70a00* is the current version. This may change if correction versions are shipped. Refer to the Release Notice.

To ensure correct execution of openUTM, you must set the UTMPATH environment variable to *utmpath*, see also [section "Starting a UTM application on Unix and Linux systems"](#).

Example

openUTM is installed under */opt/lib*

- You wish openUTM to run in 32-bit mode:
Set UTMPATH to the value */opt/lib/utm70a00/32*
- You wish openUTM to run in 64-bit mode:
Set UTMPATH to the value */opt/lib/utm70a00/64*

14.1.2 Using different socket network processes (Unix and Linux systems)

When openUTM is installed, two types of socket network processes are provided in *utmpath/ex*.

- *utmnets* resp. *utmnetsnnnn*: socket netprocesses with communication without encryption
- *utmnetssl*: socket netprocess with TLS encryption

Socket netprocesses without encryption

The network processes of the first type provide several variants. They differ mainly in the number of socket connections that may be active in parallel. The maximum number is indicated in the file name, e.g. *utmnets1024* is the socket network process for up to 1024 socket connections.

After installation, the process for up to 1024 parallel connections is used by default. The name of the currently active socket network process is always *utmnets*.

If you want to use a different socket network process, e.g. *utmnets2000* for up to 2000 socket connections, proceed as follows.

1. Terminate the UTM application
2. Copy *utmpath/ex/utmnetssl2000* to *utmpath/ex/utmnetssl*
To do this, you need the root authorization.
3. Restart the UTM application

Socket netprocesses with TLS encryption

Only one variant of the TLS network process is provided. A maximum of 1024 socket connections can be active in parallel per *utmnetssl* process.

14.1.3 Installing an openSM2 connection (Unix and Linux systems)

When openUTM is installed, the connection to openSM2 is also installed automatically. When this is done the following actions are performed:

- The script `utmsm2` is created under the directory `utmpath/shsc`. openSM2 uses this script to access the monitoring data of UTM applications.
- In Solaris and Linux systems, the script `utmsm2` is then copied to `/opt/bin`.
- In addition, the two UTM paths resulting from this installation are entered in the file `/opt/bin/utmsm2.dat`, i.e. `utm-installation-directory/utm70a00/32` and `utm-installation-directory/utm70a00/64`. If the file `/opt/bin/utmsm2.dat` does not already exist, it is first created.

If openUTM is uninstalled, these two paths are removed from the file `/opt/bin/utmsm2.dat` again. If they contain no further entries, the files `/opt/bin/utmsm2` and `/opt/bin/utmsm2.dat` are deleted.

14.2 Installing openUTM on Windows systems

Before you can create and run UTM applications on your system, you must install openUTM itself in your system.

“openUTM” is understood to be the UTM system functions (system code), C header files to create the connection program, programs for dialog terminal processes, the timer process (time control), the network processes and the tools used to create, operate and modify UTM applications.

Hardware and software requirements are listed in the Release Notice.

14.2.1 Installation of openUTM-Server (Windows systems)

The openUTM installation DVD contains a separate installation medium for the 64-bit environment.

i Note on communication component PCMX:

The openUTM installation DVD contains a separate PCMX installation medium for the 64-bit environment - PCMX-64. Please ensure that the communication component PCMX is always installed on your system.

The installation of the UTM system functions can only be done under a logon name with administrator privileges. You will need to carry out the following steps:

1. Start the program `utm-64.msi` on the openUTM installation DVD.
 - by double-clicking on Windows Explorer
 - or in the Windows command prompt by entering the command

```
msiexec /i utm-64.msi
```
2. Select the products you want to install and install PCMX, if this has not yet been installed.

PCMX is also used by other products, and a different correction version of PCMX may therefore already be installed on your computer with predefined configurations for certain network connections. However, it is recommended to use the latest version of PCMX.
3. Follow the further instructions of the installation program and select the appropriate options during the installation.

openUTM checks to make sure that the system requirements are fulfilled and there is enough disk storage capacity available. If the check fails, the installation is rejected.

If the the destination directory on the Windows computer already contains a version of openUTM then the installation procedure asks you whether the existing installation should be uninstalled or overwritten. You are recommended to uninstall the earlier version.
4. Remove the DVD from the DVD-ROM drive after the installation and reboot the system.

If you also want to compile UTM program units or link the UTM application on your Windows computer, ensure that Microsoft Visual Studio is installed.

utmpath

The directory containing the files required in order to run openUTM is referred to as **utmpath** in this manual.

During installation, *utm-installation-directory* is set up as the *utmpath* directory.

utm-installation-directory is the directory specified at installation.

To ensure that openUTM runs correctly, you must set the environment variable `UTMPATH` to the value of *utmpath*, see also [section "Starting a UTM application on Windows systems"](#).

14.2.2 User environment (Windows systems)

Administrator privileges are not required to develop applications.

Some tools (e.g. KDCDEF) must be called from a command prompt window. To start these tools using the mouse in spite of this fact, shortcuts can be set up. You will find an example of starting a UTM application via a shortcut on ["Starting with utmmain \(Windows systems\)"](#); additional examples of shortcuts can be found in the Quick Start Kit. You will also find more information on shortcuts in the Windows documentation.

14.3 Structure of the openUTM installation directory

openUTM created the following files in the *utmpath* during the installation:

File	Description
Unix, Linux and Windows systems	
applifile	General control file for all UTM applications
msgdescription	Message definition file
mtxtin	Input file for the KDCMTXT program
Unix and Linux systems	
CPIO.utmsample	utmsample application
utm.log	Log file of the installation on Unix and Linux systems
Windows systems	
AddFirewallEntries.cmd	Command file for entering the <i>utmnet/utmnets/utmnetssl</i> processes in the system's firewall. It is called when UTM is installed.
applifile.bak	Copy of the APPLIFILE
RemoveFirewallEntries.cmd	Command file for removing the <i>utmnet/utmnets/utmnetssl</i> processes from the system's firewall. It is called when UTM is uninstalled.
Uninstall.cmd	Uninstall file on Windows systems

openUTM created the following directories in the *utmpath* during the installation:

Directory	Description
Unix, Linux and Windows systems	
copy-cobol85	Copy elements for COBOL (Micro Focus compiler)
netcobol	Copy elements for COBOL (NetCOBOL compiler)
cpic	CPI-C interface
diaglst	Data structures for diagnostics
ex	Tools and programs
excel	Excel macros
include	Header files
nls	National Language Support, message catalogs

sample	openUTM sample files
shsc	Directory with UTM shell scripts and procedures
src	Source code for modifications and debugging
sys	Object libraries and object modules
tx	TX interface
upicl	Local UPIC-L directory
xatmi	XATMI for UPIC-L clients
Unix and Linux systems	
oss	OSS product
Windows systems	
log	Log files

14.4 Environment variables of a UTM application

All environment variables that can be used to control openUTM are listed in this section. They are divided into the following groups:

- [General environment variables for openUTM](#)
- [Environment variables for work processes](#)
- [Environment variables for using openssl](#)
- [Environment variables for the KDCDUMP tool](#)
- [Environment variables for the X/Open interface XATMI](#)
- [Additional environment variables for openUTM on Unix and Linux systems](#)
- [Additional environment variables for openUTM on Windows systems](#)

The meaning, range of values, default setting and UTM process that evaluates the environment variable are specified for each environment variable.

All environment variables must be set before the start of the UTM application.

i After deinstallation of openUTM on Windows systems you must check the UTMPATH and PATH environment variables and clean these up if necessary.

14.4.1 General environment variables for openUTM

UTMPATH

Meaning

Directory in which all components of openUTM and the `applifile` file are located. This environment variable must be specified in order to be able to run openUTM.

Range of values

Directory in which the files needed to run openUTM are located (*utmpath*, see ["Installing UTM system functions on Unix and Linux systems"](#)).

Default value

Unix and Linux systems: no default value. UTMPATH must always be set.

Windows systems: UTMPATH is set at installation .

Process

Evaluated by every UTM process and when a UTM tool is started.

LANG

Meaning

Language in which the UTM messages are output.

Range of values

Language code, e.g. `De_DE.646`. An NLS catalog must be available for the language.

Default value

If LANG is not set or is set incorrectly (e.g. no NLS catalog was found for the language code specified), then the messages are output in English.

Process

Evaluates in every UTM process that outputs UTM messages when the process is started.

UTM_IPC_LETTER

Meaning

Specifies the size of the data area in IPC shared memory. The data area is used to store messages that are exchanged between the processes of an application. In UTM_IPC_LETTER you specify the number of 4KB blocks that are to comprise the data area.

Range of values

Minimum: 5 (i.e. 20KB)

Default value

Depends on the number of semaphores generated.

Process

Evaluated in the first work process at the start of a UTM application.

UTM_IPC_EXTP_LETTER

Meaning

Specifies the maximum size of the data area in IPC shared memory that is available for each connection. The value specified in UTM_IPC_EXTP_LETTER is interpreted as a number of 4KB blocks (see also ["Improving performance: Changing the size of the data area in the IPC shared memory"](#)).

Range of values

Minimum: 1 (i.e. 4KB)

Default value

16 (i.e. 64KB).

Process

Evaluated in the first work process at the start of a UTM application.

UTM_REDIRECT_FILES

Meaning

Specifies whether output should be written to the existing system files *stderr* and *stdout* or their redirection destinations or not after the UTM application has been started. If UTM_REDIRECT_FILES is set to "YES", output is not written to *stdout* and *stderr*. The files are automatically switched over and output is written to the files *prefix.out.YY-MM-DD.HHMMSS* and *prefix.err.YY-MM-DD.HHMMSS* (see [section "System files stderr and stdout"](#)).

Value range

The following values are possible:

- Not set
- "YES" (Unix and Linux systems)

Default

Not set. This behavior is compatible with previous versions of openUTM.

Process

This is evaluated by the *utmmain* process when a UTM application is started.

UTM_MAIN_KILL_TIME

Meaning

Contains the time in seconds by which the *utmmain* process is delayed by calling *sleep()*:

- before a work process is started
- before all resources are deleted at the end of the application
- before the named pipe to the *utmlog* process is deleted
- after the *utmshut* process has been started. (Windows systems)

Range of values

1 through 99(sec)

Default value

1 sec

You can decrease the time it takes to terminate the UTM application or to subsequently start processes by setting `UTM_MAIN_KILL_TIME` to a smaller value.

Process

`UTM_MAIN_KILL_TIME` is evaluated when the UTM application is started in the process `utmmain`.

UTM_CORE_DUMP

Meaning

Prevents core dumps on Unix and Linux systems and mini dumps on Windows systems respectively from being created if

- a UTM dump is created in the work process, or if
- external UTM processes are terminated abnormally.

Range of values

If `UTM_CORE_DUMP` is set to "NO", no core dump or mini dump is created.

Default value

None. If not set at all or not set to "NO", a core dump or a mini dump is created in the situations described above.

Process

Evaluated in each UTM process when a core dump or a mini dump is requested.

UTM_MSG_DATE

Meaning

Prevents the date and time from being prefixed to the outputs with destinations `STDOUT` and `STDERR`.

Range of values

If the environment variable is set and contains the value "NO", neither the date nor time is prefixed to messages output to `STDOUT` and `STDERR`.

Default value

None. If the environment variable is not set or does not contain the value "NO", the date and time are prefixed to all UTM messages in order to aid diagnostics. Messages from UTM tools are exceptions; in this case, the date and time are never prefixed.

Process

`UTM_MSG_DATE` is evaluated in each UTM process when the process is started.

UTM_MSG_PID

Meaning

Prevents the PID from being prefixed to the outputs with destinations `STDOUT` and `STDERR`.

Range of values

If the environment variable is set and contains the value "NO", the PID is not prefixed to messages output to `STDOUT` and `STDERR`.

Default value

None. If the environment variable is not set or does not contain the value "NO", the PID is prefixed to all UTM messages in order to aid diagnostics.

Process

Evaluated once in the `utmmain` process when the UTM application is started.

UTMTRAC

Meaning

Optional switch to activate the dynamic trace.

Range of values

Selection of the UTM programs to be traced and the trace units. See the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems” for the syntax.

Default value

None. If it is not set, then no dynamic trace will be created.

Process

Evaluated by every UTM process when the process is started.

14.4.2 Environment variables for work processes

KDCS_C_DEBUG

Meaning

If KDCS_C_DEBUG is set, then every call of a C/C++ or COBOL program unit and every KDCS call in a C/C++ program unit is logged to *stdout*.

Range of values

If set, then logging is activated.

Default value

No logging.

Process

Evaluated in every work process of a UTM application the first time a C/C++ or COBOL program unit is called.

UTM_ABORT_WITH_EXCEPTION

Meaning

Specifies whether or not a core is created on Unix and Linux systems or the debugger is activated on Windows systems when an application crashes. UTM_ABORT_WITH_EXCEPTION is only to be used in conjunction with the start parameter STXIT=OFF.

Range of values

If set, then a core is created or the debugger is activated when an application crashes.

Default value

No core is created or the debugger is not activated when an application crashes.

Process

Evaluated in every work process when an application crashes.

PATH

Meaning

The variable only applies to Unix and Linux systems.

It specifies the path under which the shell scripts *admlp* and *utmlp* are searched for. The *admlp* script is used when printing EAM files. The *utmlp* script is used when printing in the *utmprint* printer process.

Range of values

Number of directories (in the form *directory1:directory2:...*)

Default value

If the *admlp* or *utmlp* script cannot be found in the directories specified in PATH, then the *admlp* or *utmlp* script under \$UTMPATH/*shsc* is used.

Process

Evaluated in every work process when printing EAM files and in every print process when starting the process.

14.4.3 Environment variables for using openSSL

UTM_SSL_LIBRARY

Meaning

Contains the fully qualified name of the openSSL library required for the encryption functionality and TLS connections.

Range of values

The following values are possible:

- Not set
- Specification of a file name
The filename contains the complete path for the openSSL library.
Maximum length of the file name: 300

Default value

- Variable not set
 - Unix and Linux systems: [libssl.so](#)
 - Windows systems: libeay32.dll

An attempt is made to load the openSSL library under this default name.

- Variable with specification of a file name:
An attempt is made to load the openSSL library under the specified name.

Process

The environment variable is evaluated when starting the KDCDEF utility, an utmwork and an utmnetssl process.

For using TLS communication on Windows systems an additional environment variable must be set:

UTM_SSL_LIBRARY2

Meaning

Contains the fully qualified name of the second openSSL library required for the TLS connections.

Range of values

The following values are possible:

- Not set
- Specification of a file name
The filename contains the complete path for the second openSSL library.
Maximum length of the file name: 300

Default value

- Variable not set
ssleay32.dll

An attempt is made to load the second openSSL library under this default name.

- Variable with specification of a file name:
An attempt is made to load the openSSL library under the specified name.

Process

The environment variable is evaluated when starting the utmnetssl process.

14.4.4 Environment variables for the KDCDUMP tool

EDITOR

Meaning

Contains the editor that is called by the KDCDUMP tool in the EDT command (see the openUTM manual “Messages, Debugging and Diagnostics on Unix, Linux and Windows Systems”).

Range of values

Editor program; e.g. vi, PFE32, notepad

Default value

Unix and Linux systems: vi

Windows systems: wordpad

Process

EDITOR is evaluated by KDCDUMP when the EDT command is called.

14.4.5 Environment variables for the X/Open interface XATMI

The environment variables with which you can use to control XATMI applications are listed in the following. You will find detailed descriptions of these environment variables in the openUTM manual “Creating Applications with X/Open Interfaces”.

i The environment variables for controlling traces for the X/Open interfaces (enable/disable, set trace path names) are no longer required since the traces can be enabled via the start parameters, see [section “Start parameter file of the application”](#).

XTLCF

Meaning

Contains the name of the local configuration file (LCF) used

Range of values

File name or path name that corresponds to the conventions of the operating system. If XTPALCF is used, then XTLCF may only contain a file name.

Default value

`xatmilcf` in the directory in which the application was started.

Process

XTLCF is evaluated in every work process at the start of the process.

XTPALCF

Meaning

Specifies the directory under which openUTM searches for additional descriptions of typed buffers (see also the openUTM manual “Creating Applications with X/Open Interfaces”).

Range of values

Directories specified in the following form:

Unix and Linux systems: *directory1:directory2:...*(separated by colons)

Windows systems: *directory1;directory2;...*(separated by semi-colons)

Default value

Only search in the LCF in XTLCF (or in the file `xatmilcf`, if XTLCF is not set).

Process

XTPALCF is evaluated in every work process at the start of the process.

14.4.6 Additional environment variables for openUTM on Unix and Linux systems

UTM_NO_GCORE_DUMP

Meaning

The UTM_NO_GCORE_DUMP environment variable only applies to Unix and Linux systems. It controls the creation of a gcore in the *utmcore* script.

Range of values

You can specify the following values:

- not set
- "YES"

Default value

If the environment variable is not set or does not contain the value "YES", the *utmcore* script requests a gcore.

Process

The UTM_NO_GCORE_DUMP environment variable is evaluated in the *utmcore* script by the *utmwork* process.

14.4.7 Additional environment variables for openUTM on Windows systems

USERNAME

Meaning

The variable only applies to Windows systems. It contains the UTM user ID for the dialog terminal process *utmdtp*.

Range of values

UTM user ID

Default value

Windows logon name under which *utmdtp* is started.

Process

USERNAME is evaluated in every *utmdtp* process at the start.

UTM_PIPE_TIME

Meaning

The variable only applies to Windows systems. It contains the wait time in seconds for the named pipe handling on Windows systems.

Range of values

1 - 99 (sec).

Default value

3 (sec).

Process

UTM_PIPE_TIME is evaluated in every UTM process that uses named pipes at the start of the process.

UTM_BREAK_BEFORE_DUMP

Meaning

The variable only applies to Windows systems. It defines whether or not a breakpoint is requested before each UTM dump.

Range of values

The environment variable is either defined or not defined; no value is necessary.

If the environment variable is defined, a breakpoint is requested before each UTM dump. In this case, the application run can be continued under the control of the Microsoft Visual C++ debugger in order to analyze the error. The UTM dump is created in the process (as without breakpoint).

Default value

Environment variable not defined, i.e. no breakpoint.

Process

The environment variable is evaluated in the work process before each UTM dump.

UTM_BREAK_BEFORE_KCSTRMA

Meaning

The variable only applies to Windows systems. It defines whether or not a breakpoint is requested before each application termination.

Range of values

The environment variable is either defined or not defined; no value is necessary.

If the environment variable is defined, a breakpoint is requested before each application termination. In this case, the application run can be continued under the control of the Microsoft Visual C++ debugger in order to analyze the error. A UTM dump is created in the process (as without breakpoint).

Default value

Environment variable not defined, i.e. no breakpoint.

Process

The environment variable is evaluated in the work process before each application termination.

14.5 Structure of the accounting records of openUTM

The openUTM accounting records are written in the accounting files in the ACCNT directory.

The following two record types exist:

- Accounting records (UTMA record type)
- Calculation records (UTMK record type)

The data fields of the records that contain UTM-specific information are described in this section.

These records are described in [chapter “Accounting”](#).

14.5.1 Structure of an accounting record

+00'	C'UTMA'	C' '	
+06	Name of the UTM application	C' '	
+16	Name of the UTM user	C' '	1)
+26	Sign-on time	C' '	2)
+54	Date and time of record creation	C' '	3)
+70	Accounting unit counter	C' '	4)
+83	Number of TACs called with TACUNIT>0	C' '	5)

↑
|
└─ Distance of the field in numbers of bytes (in decimal)

Comments:

1) Name of the user. In a UTM application without generated users, openUTM enters the name of the LTERM partner.

2) Sign-on time for this user (USER) to this LTERM in the form:
FRI SEPT 15 00:00:00 2000

If only asynchronous TACs were called for this USER in the current run of the UTM application, this field contains
----- .

3) Format: *yyyymmddhhmmss* (year/month/day/hour/minute/second)

4) Sum of the accounting units for this user since the last accounting record was written or since the sign-on time.

5) The fields are separated by " | ".

14.5.2 Structure of a calculation record

+00	C'UTMK'	C' '	
+06	Application name of the UTM application	C' '	
+16	Transaction code of the program unit (TAC)	C' '	
+26	CPU time in openUTM (msec)	C' '	
+36	- Reserved -	C' '	1)
+46	Length of the input message	C' '	
+53	Length of the output message	C' '	
+60	Number of asynchronous outputs	C' '	
+70	Accounting units for LTACs	C' '	2)
+80	Name of the UTM user	C' '	
+88	Name of the LTERM partner	C' '	
+96	Real time of the program unit run (in msec)	C' '	
↑			
L	- Distance of the field in numbers of bytes (in decimal)		3)

Comments:

- 1) 0 is always entered here.
- 2) See the KDCDEF statement LTAC...,LTACUNIT=
- 3) The fields are separated by " | ".

Example: Accounting phase with distributed processing via LU6.1 with LTACs

The application VTV10S on the BRANCH computer communicates with application VTV10S on the CENTRAL computer by sending LTAC calls to the application VTV10S on CENTRAL.

BRANCH computer

UTMK VTV10S	CVARL1		0		24		73		0		0	apu	
BRCV0004		50											
UTMK VTV10S	CVARL1		0		51		73		0		0	apu	
BRCV0004		50											
UTMK VTV10S	CVARL1		0		6		722		0		0	apu	
BRCV0004		10											
UTMK VTV10S	CVAR1		10		722		2205		0		50	apu	
BRCV0004		100											
UTMK VTV10S	CVAR2		20		2209		89		0		0	apu	
BRCV0004		40											
UTMA VTV10S	apu				Fri Sep 15 11:34:50 2000		20000915113453				1133		5
UTMK VTV10S	KDCBADTC		20		0		240		0		0	apu	
BRCV0004		80											
UTMA VTV10S	apu				Fri Sep 15 11:34:50 2000		20000915113453				655		0

CENTRAL computer

This application receives the LTACs sent by the application VTV10S on the BRANCH computer. The USER and LTERM names entered here correspond to the session and LPAP names.

UTMK VTV10S	PERE		40		10		80		0		0	VTV11S
LPAP1		490										
UTMK VTV10S	RT1		30		10		12		0		0	VTV12S
LPAP1		200										
UTMK VTV10S	MPF		20		10		80		0		0	VTV12S
LPAP1		240										
UTMK VTV10S	MPF		40		10		80		0		0	VTV12S
LPAP1		250										
UTMK VTV10S	RT1		30		2205		2209		0		0	VTV12S
LPAP1		300										
UTMK VTV10S	RT1		50		4400		4404		0		0	VTV12S
LPAP1		320										
UTMK VTV10S	RT1		30		10		14		0		0	VTV12S
LPAP1		220										
UTMK VTV10S	ATAC		0		5		0		0		0	VTV12S
LPAP1		170										
...												
...												
UTMK VTV10S	VPASS		7940		8		4		0		0	ap
LTP00001		84460										

14.6 Processing print output without printer control (Unix and Linux systems)

The print output for an UTM application on Unix and Linux systems is controlled in automatic mode by the printer shell script *utmlp*. If a printer is connected to an application, a separate printer process exists for this printer. If a printer and hence a printer process receives a print job, the printer process starts the printer shell script *utmlp*. The data is transferred in a pipe.

You can create *utmlp* yourself if required. The printer process evaluates the shell variable `PATH` and searches for *utmlp*. If the printer process does not find *utmlp*, it starts the printer script *utmpath/shsc/utmlp*, which is supplied with openUTM. You can change this shell script for specific applications.

For further information on the *utmlp* script, refer to openUTM manual "Generating Applications".

If an error occurs while the data is being processed in *utmlp*, the printer script terminates with an exit code not equal to zero. The printer process generates a negative print acknowledgment. openUTM then shuts down the connection to this printer process and generates UTM message K046. The printer process terminates.

All output jobs for a printer are buffered by openUTM in the message queue of the associated LTERM partner. The messages to be printed are not lost in the event of a negative print acknowledgment. They are sent to this printer the next time a connection is established.

14.7 Sample programs and sample applications

openUTM is shipped as standard with sample programs, sample procedures, and executable sample applications. These examples can be used as templates for your own application development and adapted as appropriate. You can find the description of the sample programs for administration in openUTM manual “Administering Applications”.

14.7.1 Sample programs for a publish / subscribe server

These sample programs are intended to illustrate how to implement a simple publish and subscribe service in a UTM application.

Function

A user can subscribe to a service. That user then receives or messages published as of that time in their USER queue.

The possible commands for this service are:

- help: Get help text
- subscribe: Subscribe to messages
- unsubscribe: Cancel subscription to messages
- who: Output the names of the subscribers
- publish <message>: Publish a message

The service is provided by an asynchronous service with the TAC PUBSUBA which constantly listens for jobs at the TAC queue PUBSUBMQ. Users communicate with the service over the dialog service PUPSUBD. Job confirmations are sent to the USER queue of the user and can, for instance, be read using the dialog program UPDGET (see sample programs for asynchronous processing for a UPIC client). In addition, PU can be queried in the INIT of each program unit to establish whether messages are waiting in the user's queue.

The service need only be started once by calling the TAC PUBSUBA. The open asynchronous service is then retained throughout the entire duration of the application. It is transferred to the new application by KDCUPD when a new UTM generation is performed.

If the asynchronous service terminates abnormally as the result of an error, the most recently processed job is placed in the dead letter queue.

Delivery

The programs are part of the sample application and supplied as *pubsubd.c* and *pubsuba.c*.

UTM generation

The statements for the program units in the KDCDEF run are specified as comments in the individual source files. This also applies to the statement for the TAC queue "PUBSUBMQ".

At least one GSSB must be generated (MAX GSSBS), as the service uses the GSSB "PUBSUBGB" to manage the subscribers.

If the most recently processed job is to be placed in the dead letter queue after the service is cancelled, MAX REDELIVERY = (... ,0) must be generated. If this is not done, the job remains in the job queue PUBSUBMQ.

14.7.2 Sample program for moving messages from the dead letter queue selectively

Function

The dialog program moves all messages from the dead letter queue using a specified original destination and a specified new destination. This means that a total of 16 characters are expected, i.e either two TACs or two LPAP partners or two OSI-LPAP partners, depending on the type of the original target. The program confirms the number of messages moved.

Delivery

The C program is part of the sample application and is supplied as *dadmmvsc.c*.

UTM generation

The statements for the program units in the KDCDEF run are specified as comments in the individual source files.

14.7.3 Sample programs for HTTP clients

There are four sample programs for HTTP clients available.

- HTTPECHO
- HTTPEXH
- HTTPEXT
- HTTPINF

Function

The functional details are specified as comments in the individual source files.

Delivery

The C programs are part of the sample application and are supplied as `httpecho.c`, `httpexh.c`, `httpext.c` and `httpinf.c`.

UTM generation

The statements for the program units in the KDCDEF run are specified as comments in the individual source files.

14.7.4 CPI-C sample programs

Unix and Linux systems

You will find the CPI-C sample programs on Unix and Linux systems in *utmpath/cpic/sample*.

Name	Meaning
<code>../src/kcpsam1.c</code>	C source (asynchronous part)
<code>../src/kcpsam2.c</code>	C source (synchronous part)
<code>../sys/libcpsam.a</code>	Library with objects from asynchronous part and synchronous part

Windows systems

These programs are contained in the Quick Start Kit on Windows systems, see "[openUTM Quick Start Kit on Windows systems](#)".

14.7.5 Sample procedures on Unix and Linux systems

The procedures listed below are supplied as components of the product openUTM on Unix and Linux systems, and you require them when installing or creating the application, for example. You can modify and extend the procedures in accordance with your requirements. The procedures contain comments in English.

The following sample procedures are supplied in the *utmpath/shsc* directory:

Procedure	Function
admlp	Prints the administration command output with the <code>lp</code> command
CCmainutm	Creates an object for the C++ connection
dumpstart	Used for evaluating the symbols of a UTM dump
stat2dyn	Creates a shared object library from a static library
utmlp	For processing print output
utm-c.emergency	Emergency-Skript für UTM-Cluster-Anwendungen
utm-c.failure	Failure-Skript für UTM-Cluster-Anwendungen

14.7.6 Sample procedures on Windows systems

On Windows systems, the procedures listed below are openUTM product components and are required for the operation of UTM cluster applications. You can modify and extend the procedures to meet your own requirements. The procedures contain comments in English.

The following sample procedures are supplied in the directory *utmpath*\shsc:

Procedure	Function
utm-c.emergency	Emergency script for UTM cluster applications
utm-c.failure	Failure script for UTM cluster applications

14.7.7 Sample application on Unix and Linux systems

openUTM is supplied with a sample application called *utmsample*. The file `CPIO.utmsample` of the sample application are installed in together with openUTM and are is located in the *utmpath*.

You install the sample application (including the description) under your user ID by calling the procedure *utmpath* /shsc/install.sample. From the files of the sample application, you can immediately create a simple UTM application by calling a shell procedure.

The sample procedure makes it easier to generate and start up your UTM application. From the sample application, you create an application that uses the components and interfaces (database connection e.g. Oracle, distributed processing via OSI TP or LU6.1 etc.) that your require for your application. The input file for the KDCDEF generation tool and the makefile of this application can then be used as templates for your application.

The source codes are supplied for all programs of the sample application to make it easier for you to program your own UTM programs.

14.7.8 openUTM Quick Start Kit on Windows systems

The UTM sample application inclusive the procedures are supplied in the form of a Quick Start Kit with openUTM for Windows. The Quick Start Kit requires an installed openUTM.

The Quick Start Kit contains executable sample programs, from a CPI-C client up to a UTM database application. Icons are set up for this program during the installation so that you can start the programs by clicking on the icons with the mouse. This will help you to easily learn about the openUTM functionality.

The Quick Start Kit also contains the corresponding program sources, command files and configuration files. These source files should help you when programming your own applications and can be used as templates for your own applications, for example.

The Quick Start Kit also contains documentation. The documentation contains short descriptions of the function of the programs and source files supplied.

15 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 standard defined by the National Institute of Standards and Technology (NIST) and based on the Rijndael algorithm developed at the University of Leuven (Belgium). If the AES method is used, the UPIC client generates an AES key for each session.

Apache Axis

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

Apache Tomcat

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

application 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* (eXtended Host Code Support) is used, each character set used is uniquely identified by a coded character set name (abbreviation: "CCS name" or "CCSN").

cold start

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

communication area (KDCS)

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

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

communication end point

see *transport system end point*

communication resource manager

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

configuration

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

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

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

confirmation job

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

connection bundle

see *LTERM bundle*.

connection user ID

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

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

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

contention loser

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

contention winner

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

conversation

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

conversation ID

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

CPI-C

CPI-C (**C**ommon **P**rogramming **I**nterface for **C**ommunication) is a program interface for program-to-program communication in open networks standardized by X/Open and CIW (**C**PI-C **I**mplementor's **W**orkshop).

The CPI-C implemented in openUTM complies with X/Open's CPI-C V2.0 CAE Specification. The interface is available in COBOL and C. In openUTM, CPI-C can communicate via the OSI TP, LU6.1 and UPIC protocols and with openUTM-LU62.

Cross Coupled System / XCS

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

data access control

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

data space (BS2000 systems)

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

dead letter queue

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

DES

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

dialog conversation

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

dialog job, interactive job

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

dialog message

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

dialog program

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

dialog service

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

dialog step

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

dialog terminal process (Unix , Linux and Windows systems)

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

distributed processing

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

distributed transaction

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

distributed transaction processing

Distributed processing with distributed transactions.

dynamic configuration

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

encryption level

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

event-driven service

This term has been superseded by *event service*.

event exit

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

event function

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

event service

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

filebase

UTM application filebase

On BS2000 systems, filebase is the prefix for the *KDCFILE*, the *user log file* USLOG and the *system log file* SYSLOG.

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

Functional Unit (FU)

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

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

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

generation

See *UTM generation*.

global secondary storage area

See *secondary storage area*.

hardcopy mode

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

heterogeneous link

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

Highly Integrated System Complex / HIPLEX[®]

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

HIPLEX® MSCF

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)

Provides the infrastructure and basic functions for distributed applications with HIPLEX®.

homogeneous link

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

inbound conversation (CPI-C)

See *incoming conversation*.

incoming conversation (CPI-C)

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

initial KDCFILE

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

initiator (CPI-C)

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

insert

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

inverse KDCDEF

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

IUTMDB

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

JConnect client

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

JDK

Java Development Kit

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

job

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

job complex

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

job-receiving service (KDCS)

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

job-submitting service (KDCS)

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

KDCADM

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

KDCDEF

UTM tool for the *generation* of *UTM applications*. KDCDEF uses the configuration information in the KDCDEF control statements to create the UTM objects *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 *OS/ 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 (**WebServices** 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.

16 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	Distributed 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 access 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

17 Related publications

You will find the manuals on the internet at <https://bs2manuals.ts.fujitsu.com>.

openUTM documentation

openUTM Concepts and Functions

User Guide

openUTM Programming Applications with KDCS for COBOL, C and C++

Core Manual

openUTM Generating Applications

User Guide

openUTM Using 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 Specification

Distributed Transaction Processing

The XATMI Specification

ISBN 1 85912 130 6

XML

W3C specification (www consortium)

Web page: <http://www.w3org/XML>